I A C

ISTITUTO PER LE APPLICAZIONI DEL CALCOLO

"MAURO PICONE"

CONSIGLIO NAZIONALE DELLE RICERCHE

QUADERNI

Serie III - N. 48

ELISABETTA GRAZZINI

Il linguaggio assemblativo del sistema PDP-11

ROMA

1977

I A C

ISTITUTO PER LE APPLICAZIONI DEL CALCOLO

"MAURO PICONE"

CONSIGLIO NAZIONALE DELLE RICERCHE

QUADERNI

Serie III - N. 48

ELISABETTA GRAZZINI

Il linguaggio assemblativo del sistema PDP-11

ROMA

1977

Finito di stampare nel mese di ottobre 1977

Tipo-litografia Marves
Via Mecenate, 35 - Roma - Tel. 730.061

I N D I C E

INDICE DELLE FIGURE

IL LINGUAGGIO ASSEMBLATIVO DEL SISTEMA PDP-11

E. Grazzini (*)

INTRODUZIONE

Lo scopo di questo quaderno è quello di presentare il linguaggio assemblativo dell'elaboratore DIGITAL PDP-11/4Ø.
Questo testo è stato pensato per coloro che abbiano una certa conoscenza degli elaboratori e della programmazione, possibilmente di linguaggi assemblativi, tenendo presente anche il fatto che molti Centri di Calcolo di Istituti Matematici sono in possesso di un elaboratore PDP-11.
Naturalmente questo non esclude che il testo possa essere usato per scopi didattici, se opportunamente corredato di esercitazioni. L'utente sarà, alla fine, in grado di scrivere programmi in linguaggio assemblativo che contengono istruzioni, direttive per l'assemblatore e operazioni di ingresso/uscita, come per esempio quelli riportati alle pagine 219 e 221 .
Il testo si articola in quattro parti.
Nella prima parte vengono illustrati tutti i modi di indirizzamento e le istruzioni.
La seconda parte riguarda più propriamente la programmazione in MACRO-11, con l'esposizione del formato delle frasi e delle direttive.
La terza parte illustra il sistema operativo DOS/BATCH-Monitor, ed in particolare le richieste programmate.
La quarta parte, infine contiene alcuni cenni sul sistema di interruzioni.

(*) Istituto Matematico "Ulisse Dini" - Università degli Studi di Firenze - V.le Morgagni 67/a - Firenze.

Sono stati consultati i seguenti manuali:

[1] - PDP-11/4Ø Processor Hand-book -
per le caratteristiche descritte nella prima e quarta parte;

[2] - DOS BATCH HANDBOOK -
per la seconda e terza parte e più precisamente:
- la sesta parte del manuale per le direttive descritte nella seconda parte del testo;
- il capitolo 3 della terza parte per il sistema operativo DOS/BATCH-Monitor.

Il criterio di esposizione seguito nel testo si differenzia in parte da quello dei manuali sopra citati. Per esempio i modi di indirizzamento diretto e indiretto non vengono illustrati separatamente, come in [1], ma ogni modo di indirizzamento indiretto è illustrato subito dopo l'analogo modo di indirizzamento diretto. Così, anche le istruzioni sono descritte iniziando da quelle più semplici, aumentando gradualmente la complessità delle operazioni che eseguono. Infine, le richieste programmate non sono descritte in ordine alfabetico, come in [2], ma sono raggruppate per argomento.
Ad esclusione della terza parte, che riguarda il sistema operativo DOS/BATCH-Monitor, il testo può essere usato anche da utenti che lavorano con altri sistemi operativi. In particolare, per utenti che usano il sistema operativo RSX-11M la seconda parte può, eventualmente, essere ampliata consultando il manuale "MACRO-11 Reference Manual", mentre l'ultima parte, per la sua struttura, può ancora servire come introduzione al sistema di interruzioni, che è gestito in modo più sofisticato.
Gli esempi presentati nel testo sono volutamente semplici, in quanto sono stati inseriti allo scopo di chiarire le caratteristiche del linguaggio, senza la preoccupazione di ottimizzare la programmazione.

E' opportuno osservare che la lettura della quarta parte può essere in un primo momento tralasciata. La gestione delle interruzioni è, infatti, del tutto trasparente per un comune programmatore. E' invece opportuno concentrare l'attenzione sulla prima parte ed in particolare sui modi di indirizzamento, che costituiscono, forse, la caratteristica peculiare di questo linguaggio assemblativo.
Si può infine osservare che le prime otto pagine della seconda parte possono sembrare "fuori posto", in quanto alcuni concetti compaiono già nella prima parte. Questa apparente incongruenza è dovuta alla struttura che si è voluto dare al testo. Infatti la prima parte è dedicata alla descrizione delle istruzioni del linguaggio che dipendono dall'hardware del PDP-11/4Ø, mentre nella seconda sono descritte quelle caratteristiche che dipendono dall'assemblatore, e quindi possono anche essere diverse da una versione all'altra dell'assemblatore stesso. E' quindi consigliabile iniziare la lettura dalle prime otto pagine della seconda parte e poi riprendere dalla prima parte.

PARTE PRIMA
I MODI DI INDIRIZZAMENDO E LE ISTRUZIONI

In questa prima parte sono descritte le caratteristiche del linguaggio assemblativo che dipendono dall'hardware, cioè i modi di indirizzamendo e le istruzioni.

1. CARATTERISTICHE GENERALI

La famiglia PDP-11 include parecchi modelli di unità centrali di controllo, (CPU), un grande numero di mezzi periferici e un software esteso.
Tutte le componenti del sistema e i periferici sono connessi e comunicano fra loro attraverso un unico canale, noto come UNIBUS, che permette un facile trasferimento di dati, di indirizzi e informazioni di controllo lungo 56 linee.

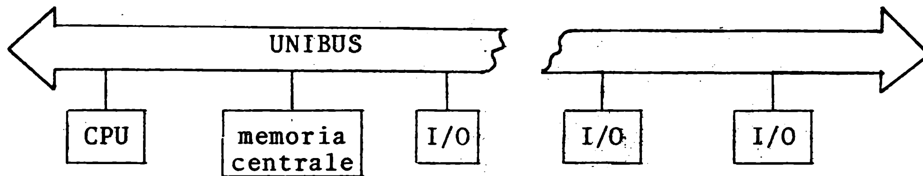


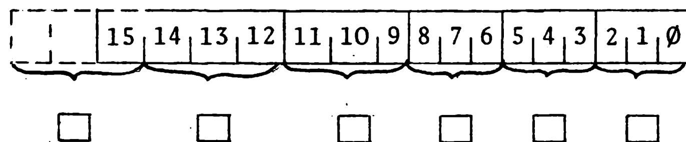
Fig.1-Schema dell'architettura del PDP-11

La forma di comunicazione è la stessa per ogni mezzo collegato con l'UNIBUS. L'unità centrale di processo usa lo stesso insieme di segnali per comunicare sia con la memoria centrale, sia con i periferici. Anche i mezzi periferici usano questo insieme di segnali per comunicare con la CPU, con la memoria centrale e con altri periferici. A ciascun mezzo, incluse le posizioni di memoria, i registri della CPU e dei periferici è assegnato un indirizzo sull'UNIBUS. Poiché le comunicazioni sull'UNIBUS sono bidirezionali e asincrone, i mezzi possono inviare, ricevere e scambiare dati con il minimo intervento dell'unità centrale; poiché è asincrono l'UNIBUS è compatibile con mezzi operanti a velocità diverse. L'unità centrale (CPU), connessa all'UNIBUS come sottosistema, controlla l'allocazione del tempo di UNIBUS per i periferici, esegue la decodifica delle istruzioni e le operazioni aritmetiche e logiche;esegue il trasferimento di dati direttamente dai mezzi di I/O alla memoria senza usare i registri dell'unità di processo;

indirizza sia uno che due operandi e manipola dati contenuti in parole di 16 bits o in bytes di 8 bits.

La CPU contiene un registro, "Processor Status Word", (PS) che descrive lo stato attuale del sistema, e 8 registri generali, numerati da Ø a 7, che possono essere usati come accumulatori, registri indice per il calcolo degli indirizzi, o come puntatori ad una pila (stack) per la memorizzazione dinamica di dati.
I registri generali si trovano agli indirizzi compresi tra 7777ØØ (indirizzo del registro Ø) e 7777Ø7 (indirizzo del registro 7), la PS si trova all'indirizzo 777776 (gli indirizzi sono espressi in ottale).
Il registro 7, R7, è usato come "program counter", (PC), contiene cioè l'indirizzo dell'istruzione successiva da eseguire ed è generalmente usato per l'indirizzamento e non come accumulatore per operazioni aritmetiche.
Il registro 6, R6, è usato automaticamente dalle istruzioni associate al collegamento dei sottoprogrammi e al servizio delle interruzioni come puntatore allo stack (hardware), cioè contiene l'indirizzo dell'ultima voce aggiunta allo stack, e per questo motivo R6 è frequentemente chiamato "SP" (stack pointer). Tuttavia ogni registro, escluso il registro 7, può essere usato come puntatore allo stack.

La parola dei sistemi PDP-11 è lunga 16 bits (due bytes di 8 bits); il suo contenuto (e indirizzo) può essere convenientemente rappresentato da un numero ottale di sei cifre:



Il bit 15, il bit più significativo (MSB) è usato come cifra più significativa del numero ottale; le altre cifre ottali sono forma-

te dai corrispondenti gruppi di 3 bits nella parola binaria.
Una parola è divisa in due bytes che costituiscono la parte alta (high byte) e la parte bassa (low byte) della parola.

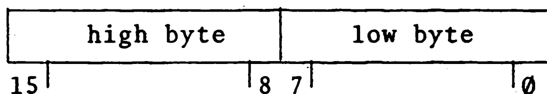


Fig.2 La parola del PDP-11

I bytes sono numerati a partire da Ø; il byte di destra di ogni parola ha indirizzo pari, il byte di sinistra ha indirizzo dispari. Le parole iniziano sempre a locazioni con indirizzo pari. Si può quindi immaginare la memoria come rappresentato in fig.1:

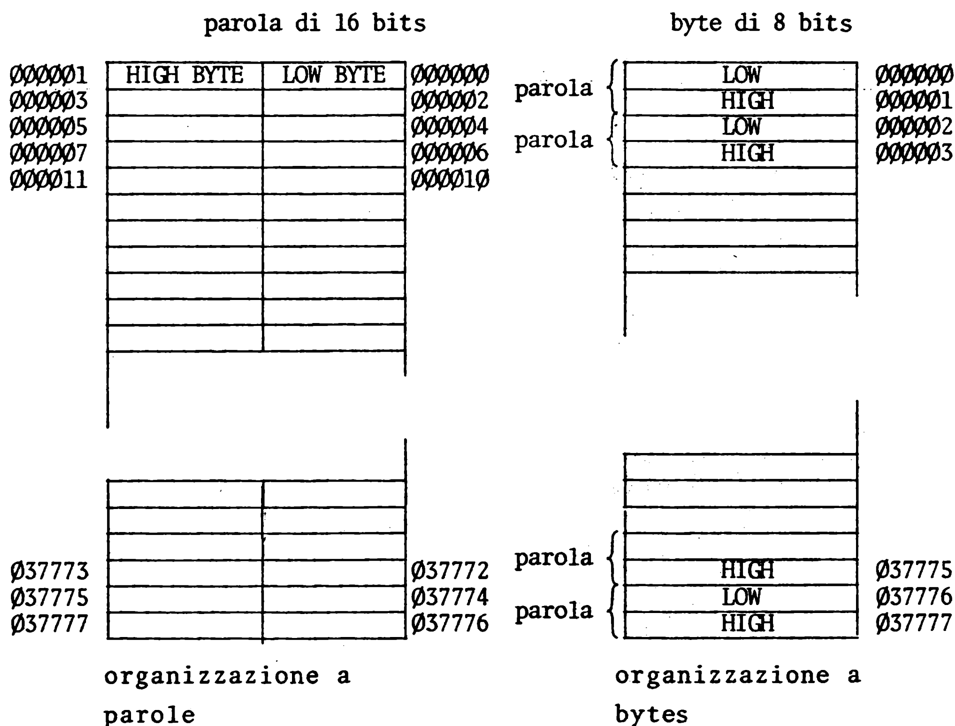


Fig.3 Organizzazione della memoria

Gli indirizzi da Ø a $37Ø_8$ sono sempre riservati e quelli fino a 777_8 sono riservati su grandi configurazioni di sistema per il trattamento delle interruzioni.
Una parola di 16 bits può indirizzare fino ad un massimo di 32k (k = 1Ø24) parole, ossia 65536 bytes.
Tuttavia le 4k posizioni di memoria con indirizzo più alto sono riservate per i registri della CPU e dei periferici: l'utente ha perciò disponibile una memoria di 28k posizioni.
Nei sistemi PDP-11/35 e 11/4Ø l'utente può utilizzare una memoria più grande di 28k, con l'opzione "Memory Management" (cfr. PDP11/4Ø - Processor Handbook), che usa un indirizzamento a 18 bits permettendo l'accesso ad una memoria attuale fino a 124k.
Se questa opzione non è usata gli indirizzi compresi tra 16ØØØØ e 177777 sono interpretati come 76ØØØØ e 777777 e indicano le 4k parole di indirizzo più alto.

2. FORMATO DELLA PROCESSOR STATUS WORD

La Processor Status word (PS), assegnata alla locazione 777776, contiene tutte le informazioni sullo stato attuale del PDP-11.

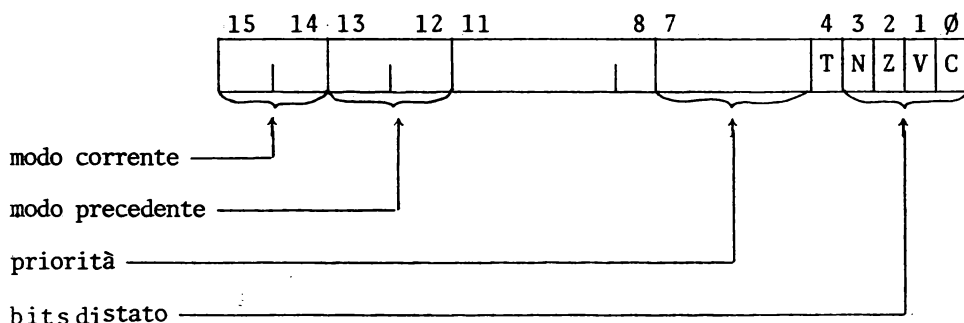


Fig. 4 Processor Status word

I bits Ø-3,bits di stato, (condition codes),contengono le informazioni sul risultato dell'ultima operazione eseguita dalla CPU. In particolare è:

Z = 1 se il risultato è Ø
N = 1 se il risultato è negativo
C = 1 se c'è un riporto dal bit più significativo
V = 1 se l'operazione ha dato luogo ad un overflow aritmetico

Il bit 4, detto "trap bit", segnala una interruzione e quando è posto uguale a 1 avviene una interruzione gestita tramite la posizione 14, che causa il caricamento di una nuova Processor Status Word (cfr. §38.e §39.)
I bits 5-7 contengono la priorità corrente; la CPU può operare a ciascuno degli otto livelli di priorità, da Ø a 7. Quando la CPU opera a livello 7, un mezzo esterno non può interromperla con una richiesta di servizio. Perché l'interruzione abbia effetto la CPU deve operare con una priorità più bassa di quella della richiesta.
I bits 8-11 non sono usati.
I bits 12-15 sono significativi soltanto quando è usata l'opzione "Memory Management" e danno informazioni sul modo di operare precedente e corrente.

3. L'INSIEME DELLE ISTRUZIONI E I TIPI DI DATI

Tutte le operazioni nel PDP-11 sono eseguite con un solo insieme di istruzioni, ossia le istruzioni usate per manipolare i dati in memoria centrale possono essere applicate anche ai dati contenuti nei registri periferici. Per esempio i dati in un registro esterno possono essere controllati o modificati direttamente dalla CPU, senza necessità di trasferirli in memoria centrale.
In questo modo viene quindi eliminata la necessità di una classe speciale di istruzioni per le operazioni di I/O.

Le istruzioni del PDP-11 offrono una vasta gamma di possibili operazioni. Esistono, infatti, istruzioni con un solo operando che permettono, ad esempio, di azzerare, incrementare, decrementare, ruotare il contenuto di una voce o di un byte; istruzioni con due operandi che, oltre al trasferimento di dati e alle operazioni aritmetiche, permettono operazioni logiche sui bits .
Il controllo del flusso di esecuzione di un programma è facilitato dalla serie di istruzioni di salto condizionato e no, per mezzo delle quali è possibile anche effettuare i collegamenti con i sottoprogrammi.
Con l'opzione "Extended Instruction Set" (EIS) (cfr.PDP-11/4Ø Processor Handbook) si possono eseguire direttamente la moltiplicazione, la divisione, gli spostamenti multipli ed operare su parole di 32 bits.
Infine con l'opzione "Floating Point" (cfr.PDP-11/4Ø Processor Handbook), che usa la EIS come prerequisito, si possono eseguire 4 istruzioni speciali per l'addizione, sottrazione, moltiplicazione, divisione di dati in virgola mobile ("floating point").

I dati alfanumerici sono rappresentati da un carattere per byte in codice ASCII (vedi appendice A).
I numeri interi in virgola fissa ("fixed point"), in precisione semplice, sono rappresentati in una voce a 16 bits con il seguente formato:

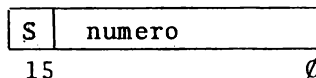


Fig.5-rappresentazione dei numeri fixed point.

ossia il bit 15 indica il segno ed è: S = Ø se il numero è positivo, S = 1 se è negativo. I numeri negativi sono rappresentati sotto forma di complemento a 2.
Si hanno quindi 15 bits per rappresentare il valore assoluto del numero; pertanto i numeri interi rappresentabili con una voce a 16 bits sono compresi tra -32768 (-2^{15}) e +32767 $(2^{15}-1)$.

In doppia precisione i dati occupano due voci; il primo bit viene sempre impiegato per rappresentare il segno e quindi si hanno 31 bits per rappresentare il numero.
I dati binari in floating point in precisione semplice occupano due voci

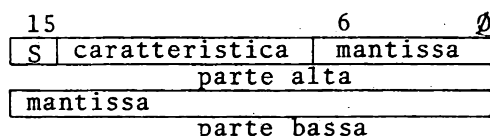


Fig.6-Rappresentazione dei numeri floating point.

Il bit 15 della prima voce rappresenta il segno della frazione; in questo caso i dati negativi non sono rappresentati nella forma di complemento a 2; i bits 7-14 della prima voce rappresentano l'esponente; gli altri 23 bits sono usati per rappresentare la frazione. Il punto decimale si considera alla sinistra, ossia davanti al bit 6 della prima voce. Poiché i numeri sono normalizzati il primo bit dopo il punto decimale è sempre 1, perciò questo bit è sottinteso e non è rappresentato direttamente. I valori che può assumere l'esponente sono compresi tra -128 e + 127; poiché viene usata una rappresentazione in eccesso a 2ØØ$_8$, ossia al valore attuale dell'esponente viene aggiunto 2ØØ$_8$(=128$_{1Ø}$), la corrispondenza fra i valori attuali e la rappresentazione codificata è la seguente:

valore attuale decimale	rappresentazione attuale	binaria
+127	377	11 111 111
⋮		
+ 1	2Ø1	1Ø ØØØ ØØ1
Ø	2ØØ	1Ø ØØØ ØØØ
- 1	177	Ø1 111 111
⋮		
-128	ØØØ	ØØ ØØØ ØØØ

Pertanto il bit 14 della prima voce non deve essere interpretato come bit di segno e gli esponenti negativi non sono rappresentati nella forma di complemento a 2. L'ordine di grandezza dei numeri rappresentabili è compreso tra $1\emptyset^{-38}$ e $1\emptyset^{+38}$.

4. FORMATO DELLE ISTRUZIONI

Le istruzioni devono essere allocate in memoria a partire da un indirizzo pari, e sono costituite da un codice operativo e dalla specificazione degli operandi che intervengono nell'operazione designata.
Le istruzioni con un solo operando hanno il seguente formato:

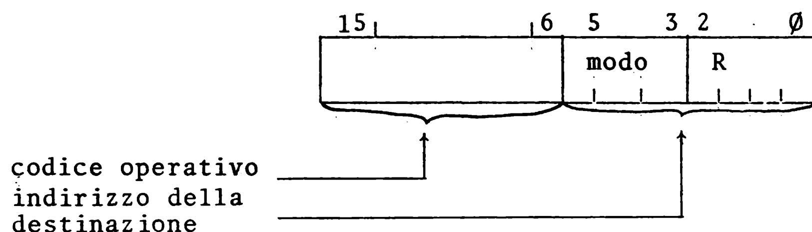


Fig.7-formato delle istruzioni con un solo operando.

I bits 6-15 specificano il codice operativo, che definisce il tipo di istruzione da eseguire. I bits Ø-5 formano un campo di sei bits, detto campo dell'indirizzo destinazione. Questo è costituito da due sottocampi:

a) i bits Ø-2 specificano a quale degli 8 registri generali si fa riferimento con questa istruzione;
b) i bits 3-5 specificano il modo in cui il registro selezionato verrà usato (modo di indirizzamento); il bit 3 è posto uguale a 1 per indicare l'indirizzamento differito (indiretto).

Le operazioni che implicano due operandi (come ad esempio l, addizione, la sottrazione, i confronti) sono eseguite da istruzioni che specificano due indirizzi. Il primo operando è chiamato

"operando sorgente", il secondo "operando destinazione". Il formato delle istruzioni con due operandi è il seguente:

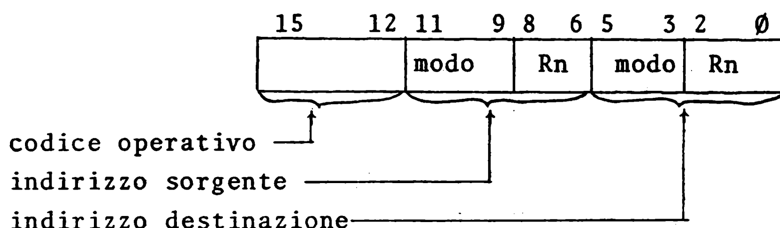


Fig.8-formato delle istruzioni con 2 operandi

I bits 12-15 specificano il codice operativo; i bits 6-11 l'indirizzo sorgente; i bits Ø-5 l'indirizzo destinazione. L'indirizzo sorgente è usato per reperire l'operando sorgente, cioè il primo operando; l'indirizzo destinazione per reperire il secondo operando e il risultato. Per esempio l'istruzione

ADD A,B

aggiunge il contenuto (operando sorgente) della locazione A al contenuto (operando destinazione) della locazione B, e dopo l'esecuzione B conterrà il risultato dell'addizione mentre il contenuto di A resta invariato.
I modi di indirizzamento e i registri usati possono essere diversi per i due operandi.

Le istruzioni di salto hanno il seguente formato:

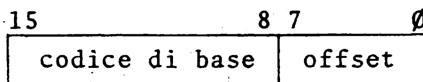


Fig.9-formato delle istruzioni di salto

Il byte di sinistra contiene il codice di base cioè il codice dell'operazione; il byte di destra contiene la distanza fra il valore attuale del "Program counter" (PC) (cfr.§1) e l'indirizzo effettivo dell'operando, ("offset").

In alcune istruzioni, come ad esempio in quelle che eseguono la moltiplicazione, la divisione, gli scorrimenti aritmetici , il salto ad un sottoprogramma, uno dei due operandi deve essere un registro generale. Tali istruzioni hanno il seguente formato:

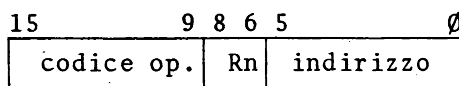


Fig.1Ø-formato delle istruzioni EIS

I bits 9-15 contengono il codice operativo; i bits 6-8 contengono il numero del registro sia che si tratti di operando sorgente sia che si tratti di operando destinazione; i bits Ø-5 contengono l'indirizzo dell'altro operando.

5. MODI DI INDIRIZZAMENTO

Al fine di comprendere come operano i vari modi di indirizzamento, è necessario ricordare come opera il "program counter". La regola chiave è la seguente:

ogni qual volta l'unità centrale usa il program counter per accedere ad una parola in memoria, il program counter è automaticamente incrementato di due dopo l'accesso alla parola .

In tal modo il PC contiene sempre l'indirizzo della parola successiva cui si deve accedere, cioè l'indirizzo dell'istruzione successiva da eseguire o della seconda o terza parola dell'istruzione corrente. Negli esempi seguenti verranno usate le seguenti istruzioni PDP-11:

codice simbolico	descrizione	codice ottale
CLR	azzera la destinazione	ØØ5ØDD
CLRB	azzera il byte della destinazione	1Ø5ØDD
INC	incrementa il contenuto della destinazione	ØØ52DD

codice simbolico	destinazione	codice ottale
INCB	incrementa il contenuto del byte destinazione	1Ø52DD
COM	sostituisce il contenuto della destinazione con il suo complemento a 1	ØØ51DD
COMB	sostituisce il contenuto del byte destinazione con il suo complemento a 1	1ØØ51DD
ADD	aggiunge l'operando sorgente all'operando destinazione; il risultato resta nell'indirizzo destinazione	Ø6SSDD
MOV	trasferisce l'operando sorgente nell'indirizzo destinazione	Ø1SSDD

dove SS indica il campo sorgente, DD indica il campo destinazione.

Per rappresentare un codice simbolico di una istruzione è usato il simbolo OPR; Rn rappresenta il nome o il numero di un registro generale.

I registri sono, di solito, indicati con i nomi RØ,R1,R2,R3,R4, R5,R6,R7; tuttavia R6 e R7 sono anche chiamati SP e PC rispettivamente.

Ricordiamo inoltre che gli indirizzi, il contenuto dei registri e delle parole, i codici delle istruzioni, sono indicati in ottale. I modi di indirizzamento descritti nei paragrafi 5.1÷5.8 usano i regitri generali RØ÷R6, mentre i modi di indirizzamento illustrati nei paragrafi 5.9÷5.12 usano esclusivamente il registro 7, ossia il PC. Quindi anche se il modo di indirizzamento è uguale non c'è ambiguità, in quanto il registro usato è sempre 7.

5.1. Indirizzamento con registro (modo Ø)

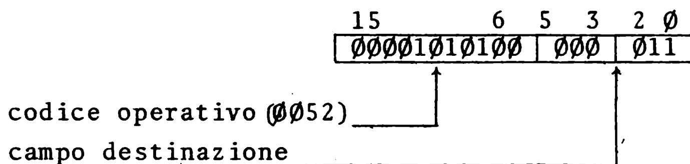
OPR Rn

Con questo modo d'indirizzamento il registro generale, indicato nell'istruzione, è usato come accumulatore e l'operando è contenuto nel registro stesso.
Poiché sono registri hardware, i registri generali operano ad alta velocità, e sono, quindi, molto vantaggiosi per operare su variabili che si usano frequentemente.

Esempi

5.1.1) INC R3

Il formato dell'istruzione è:



Come si può vedere, il sottocampo che contiene il modo di indirizzamento (bits 3-5) è zero; i bits Ø-2 contengono il numero del registro usato, ossia 3. Con questa istruzione viene aggiunto uno al contenuto del registro 3.

prima		dopo	
R3	Ø Ø Ø Ø Ø 6	R3	Ø Ø Ø Ø Ø 7

5.1.2) ADD R2, R4

Si aggiunge il contenuto del registro 2 al contenuto del registro 4; il risultato resta nel registro 4.

prima		dopo	
R2	Ø Ø Ø Ø Ø 3	R2	Ø Ø Ø Ø Ø 3
R4	Ø Ø Ø Ø Ø 5	R4	Ø Ø Ø Ø 1 Ø

5.1.3) COMB R4

Effettua il complemento a 1 del byte di destra (bits Ø-7) del registro 4, ossia ad ogni bit Ø sostituisce un bit 1, e viceversa.

prima		dopo	
R4	Ø 2 2 2 2 2	R4	Ø 2 2 1 5 5

Infatti, esprimendo in binario, abbiamo:

prima		dopo	
ØØ1ØØ1ØØ	1ØØ1ØØ1Ø	ØØ1ØØ1ØØ	Ø11Ø11Ø1
15 8	7 Ø	15 8	7 Ø

5.2. Indirizzamento con registro differito (indiretto) (modo 1)

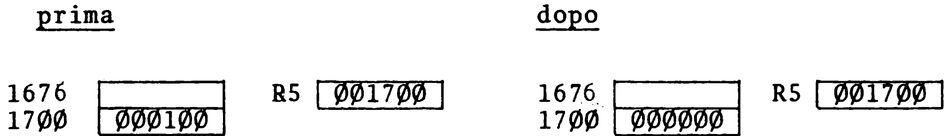
OPR @Rn oppure OPR (Rn)

Il registro non contiene l'operando, ma l'indirizzo dell'operando.

Esempi

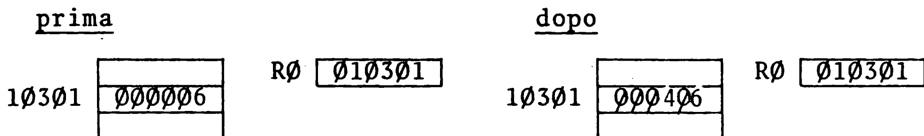
5.2.1) CLR @R5

Azzera il contenuto della posizione all'indirizzo contenuto nel registro 5.



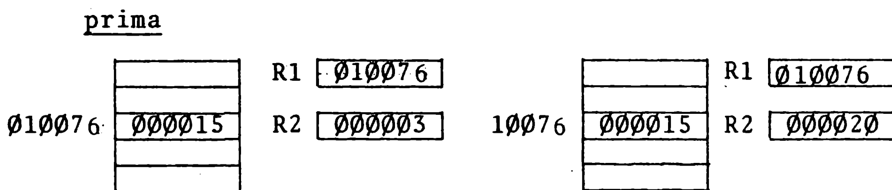
Se il contenuto del registro 5 è ØØ17ØØ, allora viene azzerata la posizione il cui indirizzo è 17ØØ; il contenuto del registro resta invariato.

5.2.2) INCB @RØ



Se il registro Ø contiene Ø1Ø3Ø1, viene incrementato il contenuto del byte all'indirizzo 1Ø3Ø1.

5.2.3) ADD (R1),R2



Se il registro 1 contiene 1ØØ76, il contenuto della posizione all'indirizzo 1ØØ76 viene aggiunto al contenuto del registro 2.

5.3. Indirizzamento con Autoincremento (modo 2).

OPR (Rn)+

Il registro è usato come puntatore a dati sequenziali; il contenuto è prima assunto come indirizzo dell'operando e poi è incrementato di uno se l'istruzione opera su bytes, oppure di due, se opera su parole puntando così alla posizione successiva.
Questo modo di indirizzamento è quindi utile per la manipolazione di dati strutturati (ad es. vettori) e di stack.

Esempi

5.3.1) CLR (R5)+

prima				dopo			
2ØØØØ	ØØ5Ø25	R5	Ø3ØØØØ	2ØØØØ	ØØ5Ø25	R5	Ø3ØØØ2
3ØØØØ	111116			3ØØØØ	ØØØØØØ		

Il contenuto di R5 è usato come indirizzo dell'operando; viene quindi azzerata la posizione 3ØØØØ e il contenuto di R5 viene incrementato di 2.

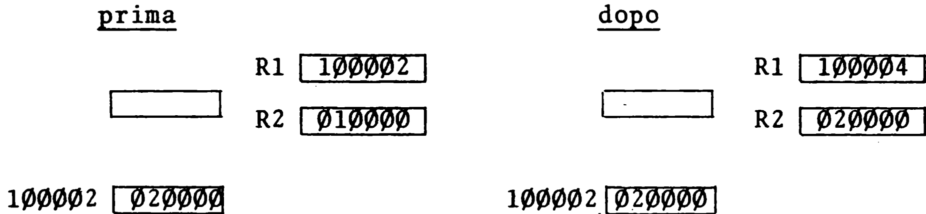
CLRB (R5)+

prima				dopo			
2ØØØØ	ØØ5Ø25	R5	Ø3ØØØØ	2ØØØØ	ØØ5Ø25	R5	Ø3ØØØ1
3ØØØØ	111116			3ØØØØ	111ØØØ		

In questo caso viene azzerato soltanto il byte di destra della posizione 3ØØØØ e il contenuto del registro viene incrementato di 1.

5.3.3)

MOV (R1)+,R2



Il contenuto della posizione 1ØØØØ2 viene trasferito nel registro 2; il contenuto di R1 è incrementato di 2.

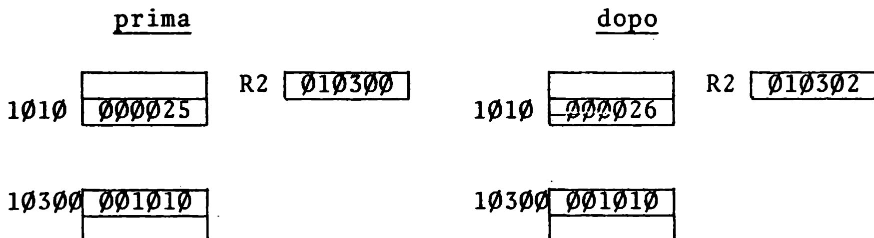
5.4. <u>Indirizzamento differito con Autoincremento</u> (modo 3).

OPR @(Rn)+

Il registro è prima usato come puntatore ad una parola contenente l'indirizzo dell'operando, poi incrementato, sempre di 2.

<u>Esempi</u>

5.4.1) INC @(R2)+



Il contenuto della posizione 1Ø3ØØ è assunto come indirizzo dell'operando; viene pertanto incrementato di 1 il contenuto della posizione 1Ø1Ø; il contenuto del registro 2 è poi incrementato

di due.

5.4.2) ADD @(RØ)+,R1

prima				dopo			
11Ø	ØØØØ1Ø	RØ	ØØ111Ø	11Ø	ØØØØ1Ø	RØ	ØØ1112
		R1	ØØØØØ1			R1	ØØØØ11
111Ø	ØØØ11Ø			111Ø	ØØØ11Ø		

Il contenuto della posizione 111Ø è assunto come indirizzo dell'operando; il contenuto della posizione Ø11Ø è aggiunto al contenuto di R1 e RØ viene incrementato di 2.

5.5. Indirizzamento con autodecremento (modo 4)

OPR -(Rn)

Il contenuto del registro è decrementato (di due per le istruzioni che operano su parole, di uno per quelle che operano su bytes) e poi usato come indirizzo dell'operando.

Esempi

5.5.1) INC -(RØ)

prima				dopo			
1ØØØ	ØØ524Ø	RØ	Ø17776	1ØØØ	ØØ524Ø	RØ	Ø17774
17774	ØØØØØØ			17774	ØØØØØ1		

Il contenuto di RØ è decrementato di due; quindi viene incrementato il contenuto della locazione all'indirizzo 17774

5.5.2) INCB - (RØ)

1ØØØ	Ø1524Ø	RØ	Ø17776		1ØØØ	Ø1524Ø	RØ	Ø17775
17775	ØØØ ØØØ	17774			17775	ØØØ 4ØØ	17774 17776	

Il contenuto di RØ è decrementato di 1; viene quindi incrementato di 1 il byte all'indirizzo 17775, ossia il byte di sinistra della parola all'indirizzo 17774.

5.5.3) ADD -(R3),RØ

prima				dopo			
1ØØ2Ø	Ø643ØØ	ØØØØ2Ø	RØ	1ØØ2Ø	Ø643ØØ	ØØØØ7Ø	RØ
17774	ØØØØ5Ø	Ø17776	R3	17774	ØØØØ5Ø	Ø17774	R3
17776				17776			

Il contenuto di R3 è decrementato di 2; quindi il contenuto della posizione all'indirizzo 17774 è aggiunto al contenuto di RØ. La possibilità di indirizzare con l'autoincremento e l'autodecremento, facilita le operazioni su stack sia hardware che software. Ricordando, infatti, che uno stack inizia alla locazione con indirizzo più alto ad esso riservata e si espande linearmente verso il basso, le voci possono essere aggiunte usando l'indirizzamento con l'autodecremento, e tolte con l'autoincremento. Supponiamo ad esempio di voler salvare il contenuto dei registri generali RØ e R1, ponendoli nello stack hardware. Ciò si ottiene con le istruzioni:

```
MOV RØ,-(SP)

MOV R1,-(SP)
```

Infatti SP contiene l'indirizzo di memoria dove è memorizzata l'ultima voce nello stack, quindi decrementandolo punterà alla prima posizione libera nello stack, dove verrà posto il contenuto di RØ ; analogamente per memorizzare R1.

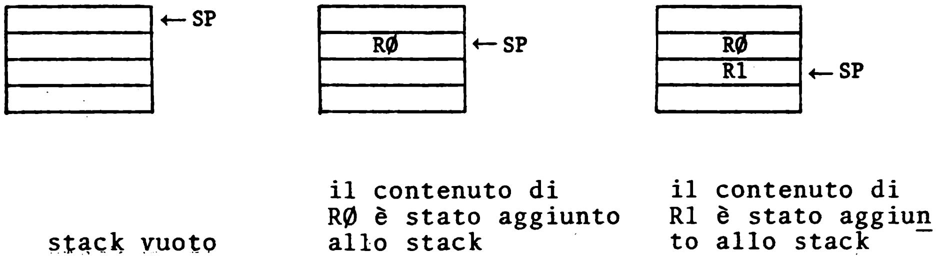


Fig.11-Esempio di memorizzazione nello stack.

Al momento di recuperare i valori di RØ e R1 potremo usare l'indirizzamento con l'autoincremento.

```
MOV  (SP)+,R1

MOV  (SP)+,RØ
```

SP contiene infatti l'indirizzo della posizione che contiene R1; dopo l'esecuzione della prima istruzione MOV il contenuto di SP viene incrementato, puntando così alla posizione contenente RØ



Fig.12-Esempi di recupero dallo stack

Il puntatore allo stack punta all'ultima posizione usata, e si sottintende che la posizione successiva (cioè quella con indirizzo più basso) sia libera.

5.6. Indirizzamento differito con Autodecremento (modo 5).

OPR @-(Rn)

Il registro è decrementato (sempre di 2) e poi usato come puntatore ad una parola contenente l'indirizzo dell'operando

Esempi

5.6.1) COM @-(RØ)

prima

Ø12345	1Ø1ØØ	RØ	Ø1Ø776
Ø1Ø1ØØ	1Ø774		
	1Ø776		

dopo

165432	1ØØ1ØØ	RØ	Ø1Ø774
Ø1Ø1ØØ	1Ø774		
	1Ø776		

Il contenuto di RØ viene decrementato di 2; l'indirizzo dell'operando è quindi il contenuto della posizione 1Ø774; viene quindi fatto il complemento a 1 della locazione all'indirizzo 1Ø1ØØ

5.6.2) MOV @-(R1),RØ

prima

1Ø1Ø2	ØØØØ7Ø	RØ	ØØØØØ1
		R1	Ø17774
17772	Ø1Ø1Ø2		
17774			

dopo

1Ø1Ø2	ØØØØ7Ø	RØ	ØØØØ7Ø
			Ø17772
17772	Ø1Ø1Ø2		
17774			

Il contenuto di R2 è decrementato di 2; il contenuto della posizione all'indirizzo 1Ø1Ø2 è sostituito al valore di RØ.

5.7. Indirizzamento con Indice (modo 6)

OPR X(Rn)

Il valore dell'espressione X è memorizzato nella seconda o terza parola seguente l'istruzione.L'indirizzo effettivo dell'operando è calcolato aggiungendo al valore dell'espressione X il contenuto del registro Rn.
In tal modo è quindi possibile un accesso casuale a elementi di dati strutturati: X è detta la base per calcolare l'indirizzo.

Esempi

5.7.1) CLR 2ØØ(R4)

prima

1Ø2Ø	ØØ5Ø64	R4	ØØ1ØØØ
1Ø22	ØØØ2ØØ		
1Ø24			
12ØØ	177777		

dopo

1Ø2Ø	ØØ5Ø64	R4	ØØ1ØØØ
1Ø22	ØØØ2ØØ		
1Ø24			
12ØØ	ØØØØØØ		

Se l'istruzione (codice ottale: ØØ5Ø64) si trova alla posizione 1Ø2Ø, nella posizione 1Ø22 si trova il valore dell'espressione indice, ossia 2ØØ. L'indirizzo dell'operando si calcola aggiungendo 2ØØ al contenuto di R4; l'operando si trova quindi alla locazione con indirizzo 12ØØ, che viene pertanto azzerata.

5.7.2) COMB 2ØØ(R1)

prima				dopo			
1Ø2Ø	1Ø5Ø61	R1	Ø17777	1Ø2Ø	Ø15Ø61	R1	Ø17777
1Ø22	ØØØ2ØØ			1Ø22	ØØØ2ØØ		
2Ø176	Ø11 ¦ ØØØ			2Ø176	1664ØØ		
2Ø2ØØ				2Ø2ØØ			

L'indirizzo effettivo dell'operando si calcola aggiungendo 2ØØ al contenuto di R1; viene fatto quindi il complemento a 1 del byte all'indirizzo 2Ø177.

5.7.3) ADD 3Ø(R2),2Ø(R5)

prima				dopo			
1Ø2Ø	Ø66265	R2	ØØ11ØØ	1Ø2Ø	Ø62265	R2	ØØ11ØØ
1Ø22	ØØØØ3Ø			1Ø22	ØØØØ3Ø		
1Ø24	ØØØØ2Ø			1Ø24	ØØØØ2Ø		
		R5	ØØ2ØØØ			R5	ØØ2ØØØ
113Ø	ØØØØØ1			113Ø	ØØØØØ1		
2Ø2Ø	ØØØØØ1			2Ø2Ø	ØØØØØ2		

Il contenuto della posizione che è determinata aggiungendo 3Ø al contenuto di R2 è aggiunto al contenuto della posizione determinata aggiungendo 2Ø al contenuto di R5. In quest'ultima posizione è memorizzato il risultato.

5.8. Indirizzamento con Indice differito(modo 7)

OPR @X(Rn)

Il valore di X e il contenuto di Rn sono sommati; il totale è usato come puntatore ad una parola contenente l'indirizzo dell'operando. Né X né il contenuto di Rn sono modificati.

Esempi

5.8.1) ADD @1ØØØ(R2),R1

prima

1Ø2Ø	Ø672Ø1	R1	ØØ1234
1Ø22	ØØ1ØØØ		
1Ø24		R2	ØØØ1ØØ
1Ø5Ø	ØØØØØ2		
11ØØ	ØØ1Ø5Ø		

dopo

1Ø2Ø	Ø672Ø1	R1	ØØ1236
1Ø22	ØØ1ØØØ		
1Ø24		R2	ØØØ1ØØ
1Ø5Ø	ØØØØØ2		
11ØØ	ØØ1Ø5Ø		

1ØØØ e il contenuto di R2 sono sommati per ottenere l'indirizzo della posizione che contiene l'indirizzo effettivo dell'operando; l'operando si trova quindi alla posizione 1Ø5Ø; viene quindi aggiunto 2 al contenuto di R1.

5.8.2) CLR @14(R4)

prima

114	ØØ2ØØØ	R4	ØØØ1ØØ
2ØØØ	Ø16754		

dopo

114	ØØ2ØØØ	R4	ØØØ1ØØ
2ØØØ	ØØØØØØ		

La posizione 114 contiene 2ØØØ; viene quindi azzerata la posi-

zione all'indirizzo 2ØØØ.

5.9. Indirizzamento immediato, uso del registro 7 (modo 2)

OPR #n,

L'operando stesso è memorizzato come seconda o terza voce dell'istruzione. Questo modo di indirizzamento è assemblato come un autoincremento del registro 7, il PC.

Esempi

5.9.1) MOV #1ØØ,R3

L'istruzione è assemblata in due parole di cui la prima contiene il codice dell'istruzione (codice operativo, modo di indirizzamento dell'operando sorgente e dell'operando destinazione), la seconda contiene la costante 1ØØ. Subito prima di accedere all'istruzione il PC punta alla prima parola; dopo l'accesso alla prima voce il PC è incrementato di 2 e contiene quindi l'indirizzo della seconda voce.

prima

Ø127Ø3	←PC	R3	ØØØØØØ
ØØØ1ØØ			

dopo

Ø127Ø3		R3	ØØØ1ØØ
ØØØ1ØØ			
	←PC		

Poiché l'operando sorgente è indirizzato con l'autoincremento del registro 7, il PC è usato come puntatore all'operando sor-

gente (la seconda parola dell'istruzione) prima di essere incrementato di due per puntare all'istruzione successiva.
Il valore 1ØØ viene trasferito nel registro R3.

Esempi

5.9.2) ADD #1Ø,RØ

prima

1Ø2Ø	Ø627ØØ	←PC	RØ	ØØØØ2Ø
1Ø22	ØØØØ1Ø			
1Ø24				

dopo

1Ø2Ø	Ø627ØØ		RØ	ØØØØ3Ø
1Ø22	ØØØØ1Ø			
1Ø24		←PC		

Il valore 1Ø è posto nella seconda voce dell'istruzione; al momento dell'esecuzione il PC contiene l'indirizzo 1Ø22 e quindi, essendo assemblato come autoincremento, si ha che il valore 1Ø è aggiunto al registro Ø, e il PC è incrementato di 2.

5.10 Indirizzamento Assoluto,uso del registro 7(modo 3).

OPR @#A

Questo modo è equivalente all'indirizzamento immediato differito. Il contenuto della posizione che segue l'istruzione è interpretato come l'indirizzo assoluto dell'operando.
Il modo di indirizzamento assoluto è assemblato, come un autoincremento differito del registro 7.

Esempi

5.1Ø.1) CLR @#11ØØ

prima			dopo	
2Ø	ØØ5Ø37	←PC		ØØ5Ø37
22	ØØ11ØØ			ØØ11ØØ
11ØØ	177777		11ØØ	ØØØØØØ
11Ø2			11Ø2	

Azzera il contenuto della posizione con indirizzo assoluto 11ØØ.

5.10.2) ADD @ #2ØØØ,R3

prima					dopo				
2Ø	Ø637Ø3	←PC	R3	ØØØ5ØØ	2Ø	Ø637Ø3		R3	ØØ1ØØØ
22	ØØ2ØØØ				22	ØØ2ØØØ			
24							←PC		
2ØØØ	ØØØ3ØØ								

Il contenuto della locazione all'indirizzo 2ØØØ è aggiunto al contenuto di R3.

5.11. Indirizzamento relativo,uso del registro 7(modo 6)

OPR A

Il modo di indirizzamento relativo è assemblato come un modo indice che usa il PC come registro indice. La base per il calcolo dell'indirizzo, che è memorizzata nella seconda o terza parola dell'istruzione non è l'indirizzo dell'operando, ma il numero che aggiunto al valore corrente del PC diventa l'indirizzo dell'operando.

Esempio

5.11.1) INC A

prima			dopo		
1020	005267	← PC	1020	005267	
1022	000054		1022	000054	
1024			1024		← PC
1100	000000		1100	000001	

Supponiamo che la posizione A abbia indirizzo 1100, e che l'istruzione sia assemblata alla posizione assoluta 1020. Dopo l'accesso all'istruzione INC il PC punta alla posizione 1022. Poiché il modo di indirizzamento dell'operando sorgente è quello indice con il registro 7, si accede alla parola puntata dal PC, che quindi viene incrementato di 2. Pertanto il PC punta alla locazione 1024. L'indirizzo effettivo dell'operando è quindi calcolato aggiungendo la base al registro 7, cioè:

1024 + 54 = 1100

Si procede quindi incrementando di 1 il contenuto della locazione 1100.

Questo modo di indirizzamento è chiamato relativo perché l'indirizzo dell'operando è calcolato relativamente al valore corrente del PC. La base è la distanza (o "offset"), in bytes, fra l'operando e il PC corrente.

5.12. Indirizzamento relativo differito, uso del registro 7 (modo 7)

OPR @A

In questo modo di indirizzamento, analogo all'indirizzamento relativo, la seconda parola dell'istruzione, sommata al PC dà l'indirizzo dell'indirizzo dell'operando, invece dell'indirizzo dell'operando.

Esempio

5.12.1) CLR @A

prima

1Ø2Ø	ØØ5Ø77	←PC
1Ø22	ØØØØ2Ø	
1Ø24		
1Ø44	Ø1Ø1ØØ	
1Ø1ØØ	1ØØØØ1	

dopo

1Ø2Ø	ØØ5Ø77	
1Ø22	ØØØØ2Ø	
1Ø24		←PC
1Ø44	Ø1Ø1ØØ	
1Ø1ØØ	ØØØØØØ	

La seconda voce dell'istruzione è aggiunta al PC per ottenere l'indirizzo della posizione che contiene l'indirizzo effettivo dell'operando. L'indirizzo effettivo è quindi 1Ø1ØØ.
Viene azzerata la posizione il cui indirizzo è contenuto in A.

5.12.2) MOV @X,RØ

Trasferisce il contenuto della posizione il cui indirizzo è in X nel registro Ø.

Il modo di indirizzamento assoluto differisce da quello relativo, in quanto la seconda o terza parola dell'istruzione contengono l'indirizzo dell'operando invece della distanza relativa fra l'operando e il PC. Così l'istruzione:

CLR @#1ØØ

azzera la locazione assoluta 1ØØ, anche se l'istruzione è mossa dal punto in cui è stata assemblata.

6. LE ISTRUZIONI

Il sistema PDP-11 include un insieme di istruzioni che manipolano, come operandi, bytes.Le istruzioni che operano sui bytes indirizzandoli con l'autoincremento o con l'autodecremento fanno si che il registro specificato sia modificato di uno per puntare al successivo byte ; usando l'indirizzamento con registro si accede al byte di destra del registro specificato. Nelle istruzioni che operano su bytes il bit 15 è uguale a 1.
La descrizione di ciascuna istruzione include il codice simbolico, il codice ottale, una descrizione della sua esecuzione e dell'effetto sui bits N,Z,V,C (condition codes).Per le istruzioni che operano su bytes,è indicato anche il relativo codice simbolico.
Nella descrizione del codice ottale la cifra più significativa (che può assumere valore Ø o 1) è indicata con un * :

$$* = \begin{cases} Ø \text{ per parola} \\ 1 \text{ per byte} \end{cases}$$

Inoltre ( ) indica il contenuto, ossia (R1) indica il contenuto di R1.

6.1. Le istruzioni generali

► CLR clear destination

► CLRB

codice ottale *Ø5ØDD

Il contenuto della destinazione (parola o byte) è sostituito con tutti zero. Il bit Z è posto uguale 1, gli altri sono azzerati.

Esempi

CLR R1

prima	dopo
(R1) = 177777	(R1) = ØØØØØØ
N Z V C	N Z V C
1 1 1 1	Ø 1 Ø Ø

CLRB R1

prima	dopo
(R1) = 177777	(R1) = 1774ØØ
N Z V C	N Z V C
1 1 1 1	Ø 1 Ø Ø

► COM

complement destination

► COMB

---

codice ottale *Ø51DD

Sostituisce il contenuto dell'indirizzo destinazione con il corrispondente complemento logico, ossia ciascun bit Ø è posto uguale a 1 e ciascun bit 1 è azzerato. Inoltre:

N = 1 se il bit più significativo del risultato è = 1; altrimenti è azzerato.

Z = 1 se il risultato è ∅; altrimenti è azzerato.

V = ∅

C = 1

Esempio

COM R∅

prima	dopo
(R∅) = ∅13333	(R∅) = 164444
R∅ ∅∅∅1∅11∅11∅11∅11	R∅ 111∅1∅∅1∅∅1∅∅1∅∅
N Z V C	N Z V C
∅ 1 1 ∅	1 ∅ ∅ 1

► INC

increment destination

► INCB

codice ottale *∅52DD

Aggiunge uno al contenuto della destinazione.

N = 1 se il risultato è < ∅ ; altrimenti è azzerato.

Z = 1 se il risultato è ∅ ; altrimenti è azzerato.

V = 1 se la destinazione contiene ∅77777;altrimenti è azzerato.

C : inalterato

Esempi

INC R2

prima	dopo
(R2) = ØØØ333	(R2) = ØØØ334
N Z V C	N Z V C
Ø Ø Ø Ø	Ø Ø Ø Ø

► DEC
decrement destination
► DECB

codice ottale *Ø53DD

Sottrae 1 al contenuto della destinazione.

N = 1 se il risultato è < Ø ; altrimenti è azzerato.

Z = 1 se il risultato è Ø ; altrimenti è azzerato.

V = 1 se la destinazione contiene $1ØØØØØ_8$; altrimenti è azzerato.

C : inalterato

Esempio

DEC R5

prima	dopo
(R5) = ØØØØØ1	(R5) = ØØØØØØ
N Z V C	N Z V C
1 Ø Ø Ø	Ø 1 Ø Ø

► NEG

negate destination

► NEGB

---

codice ottale *Ø54DD

Sostituisce il contenuto della destinazione con il suo complemento a 2. E' da tener presente che il numero $1ØØØØØ_8$ resta invariato, in quanto è il più piccolo numero negativo.

N = 1 se il risultato è <Ø; altrimenti è azzerato
Z = 1 se il risultato è Ø; altrimenti è azzerato
V = 1 se il risultato è $1ØØØØØ_8$; altrimenti è azzerato
C = Ø se il risultato è Ø; altrimenti è azzerato

Esempio

NEG RØ

prima	dopo
(RØ) = ØØØØ1Ø	(RØ) = 17777Ø
N Z V C	N Z V C
Ø Ø Ø Ø	1 Ø Ø 1

► TST

test destination

► TSTB

---

codice ottale *Ø57DD

Questa istruzione opera solo sui bits Ø-3 della PS. Mette a 1 i bits N e Z in accordo al contenuto dell'indirizzo destinazione.

In particolare:

N = 1 se il contenuto è < Ø ; altrimenti è posto = Ø

Z = 1 se il risultato è Ø ; " " " " "

V : azzerato

C : azzerato.

Il contenuto della destinazione resta invariato.

Esempi

TST R1

prima	dopo
(R1) = Ø1234Ø	(R1) = Ø1234Ø
N Z V C	N Z V C
Ø Ø 1 1	Ø Ø Ø Ø

TST R2

prima	dopo
(R2) = 117776	(R2) = 117776
N Z V C	N Z V C
Ø Ø 1 1	1 Ø Ø Ø

TSTB R2

prima	dopo
(R2) = 117Ø76	(R2) = 117Ø76
N Z V C	N Z V C
Ø Ø 1 1	Ø Ø Ø Ø

Controlla il bit 7 di R2 e in base al suo valore mette a Ø il bit N.

TSTB R3

prima	dopo
(R3) = 11ØØØØ	(R3) = 11ØØØØ
N Z V C	N Z V C
Ø Ø 1 1	Ø 1 Ø Ø

Con questa istruzione vengono messi a zero i bits V e C.

► MOV

move source to destination

► MOVB

---

codice ottale *1SSDD

Trasferisce l'operando sorgente nella posizione indicata dall'indirizzo destinazione. Il contenuto precedente della destinazione è perduto, mentre quello sorgente resta inalterato.
L'istruzione MOVB applicata ad un registro estende il bit più significativo del byte di destra. Negli altri casi MOVB opera sui bytes esattamente come MOV opera sulle parole.

Esempi

MOV #2Ø,RØ

Trasferisce il numero 2Ø (ottale) nel registro Ø

prima	dopo
(RØ) = Ø1ØØ17	(RØ) = ØØØØ2Ø

MOVB #2Ø,R1

Trasferisce il numero $2Ø_8$ nel byte di destra del registro 1

prima	dopo
(R1) = ØØØØØØ	(R1) = ØØØØ2Ø

MOVB #-1,RØ

(RØ) = ØØØØØØ	(RØ) = 177777

MOV R1,R3

Trasferisce il contenuto di R1 in R3.

prima	dopo
(R1) = ØØ1777	(R1) = ØØ1777
(R3) = ØØ1677	(R3) = ØØ1777

L'effetto sui bits condition codes è il seguente:

N = 1 se il contenuto dell'operando sorgente è < Ø; altrimenti=Ø

Z = 1 " " " " " " " = Ø; " "

V = Ø

C : non interessato

► CMP

compare

► CMPB

codice ottale *2SSDD

Confronta l'operando sorgente con l'operando destinazione, cioè sottrae dal contenuto dell'indirizzo sorgente il contenuto dell'indirizzo destinazione. [(src)-(dst)], e mette a 1 o a Ø i condition codes in base al risultato del confronto. In particolare:

N = 1 se il risultato è < Ø ; altrimenti è posto = Ø

Z = 1 se il risultato è Ø ; altrimenti è Ø

V = 1 se avviene un overflow aritmetico, cioè gli operandi erano di segno opposto e il segno della destinazione era lo stesso del risultato; altrimenti è Ø

C = Ø se vi è un riporto dal bit più significativo del risultato; altrimenti è 1.

Entrambi gli operandi restano invariati; l'unico effetto è quello di cambiare il valore dei bits N,Z,V,C, per cui questa istruzione è generalmente seguita da una istruzione di salto condizionato.

► ADD add

codice ottale Ø6SSDD

Somma l'operando sorgente all'operando destinazione e lascia il risultato nell'indirizzo destinazione. Il contenuto originale della destinazione è perduto, mentre il contenuto dell'operando sorgente resta invariato.

N = 1 se il risultato è < Ø ; altrimenti è azzerato

Z = 1 se il risultato è Ø ; altrimenti è azzerato

V = 1 se vi è un overflow aritmetico come risultato dell'operazione; cioè entrambi gli operandi hanno lo stesso segno e il risultato è di segno opposto; altrimenti è azzerato.

C = 1 se vi è un riporto dal bit più significativo del risultato; altrimenti è azzerato.

Esempio

ADD R1,R2

Aggiunge al contenuto di R2 il contenuto di R1.

prima	dopo
(R1) = ØØØØ1Ø	(R1) = ØØØØ1Ø
(R2) = ØØØØØ1	(R2) = ØØØØ11
N Z V C	N Z V C
Ø 1 Ø Ø	Ø Ø Ø Ø

► SUB subtract

codice ottale 16SSDD

Sottrae l'operando sorgente dall'operando destinazione e lascia il risultato nell'indirizzo destinazione. Il contenuto originale della destinazione è perso, mentre quello dell'operande sorgente resta invariato.

N = 1 se il risultato è < Ø ; altrimenti è azzerato

Z = 1 se il risultato è Ø ; altrimenti è azzerato

V = 1 se vi è un overflow aritmetico come risultato dell'operazione; cioè gli operandi erano di segno opposto e il segno dell'operando sorgente è lo stesso del risultato; altrimenti è azzerato

C = Ø se vi è un riporto dal bit più significativo del risultato; altrimenti è 1.

Esempio

SUB R1,R2

Sottrae al contenuto di R2 il contenuto di R1

prima	dopo
(R1) = Ø11111	(R1) = Ø11111
(R2) = Ø12345	(R2) = ØØ1234
N Z V C	N Z V C
1 1 1 1	Ø Ø Ø Ø

6.2. Le istruzioni di salto

Queste istruzioni causano un salto alla locazione definita dalla somma dell'offset (moltiplicato per 2) e il contenuto corrente del PC se:

a) è una istruzione di salto incondizionato

b) è una istruzione di salto condizionato e le condizioni sono soddisfatte, dopo aver controllato i condition codes.

L'indirizzo del salto è calcolato come segue:

1) il segno dell'offset è esteso ai bits 8-15

2) il risultato è moltiplicato per 2;

3) il risultato è aggiunto al PC per ottenere l'indirizzo finale del salto.

L'assemblatore (Assembler) esegue l'operazione inversa per determinare la distanza in bytes,dall'indirizzo specificato. E' da tener presente che quando l'offset è aggiunto al PC, il PC punta alla parola seguente l'istruzione di salto.
Il bit 7 dell'istruzione di salto è il segno dell'offset; se è = 1 l'offset è positivo e il salto è fatto *in avanti* .
L'offset di 8 bits permette un salto *indietro* di 200_8 parole dal valore corrente del PC (400_8 bytes) e in avanti di 177_8 parole (376_8 bytes). La sintassi delle istruzioni di salto è:

Bxx loc

dove "Bxx" è il nome simbolico dell'istruzione e "loc" è l'indirizzo dell'istruzione a cui bisogna andare. L'assemblatore segnala un errore nella istruzione se si supera il rango permesso per il salto. Le istruzioni di salto non hanno nessun effetto sui condition codes.

6.2.1. <u>Le istruzioni di salto incondizionato</u>

► BR branch (unconditional)

codice ottale 000400 più offset

Con questa istruzione si può trasferire il controllo del programma all'interno di un rango da -128 a +127 parole.

Esempio

```
            MOV   #BUFF,RØ
            MOV   #BUFF1,R1
      A :   MOV   (RO)+,(R1)+
        B { :
            BR    A
```

Si trasferisce in RØ l'indirizzo di una memoria di transito ("buffer"), BUFF, e in R1 l'indirizzo di un altro buffer, BUFF1; l'istruzione con etichetta (1) A trasferisce i dati del primo buffer nel secondo; dopo la sequenza di istruzioni B, l'istruzione BR fa ritornare il controllo all'istruzione A.

► JMP jump

codice ottale ØØØ1DD

In questa istruzione il byte di destra non contiene l'offset ma l'indirizzo della destinazione. Con questa istruzione il controllo può essere trasferito ad una posizione qualsiasi in memoria ed è compatibile con tutti i modi di indirizzamento, escluso il modo Ø (indirizzamento con registro). L'indirizzamento indiretto con registro è invece possibile, ed in questo caso il controllo del programma è trasferito all'indirizzo contenuto nel registro specificato.
Con questa istruzione è quindi possibile trasferire il controllo a indirizzi variabili durante l'esecuzione del programma, tenendo presente che le istruzioni devono iniziare ad un indirizzo pari.

---

(1) L'etichetta è un nome di al più 6 caratteri che rappresenta simbolicamente un indirizzo di memoria. (cfr.§12.1)

Esempio

```
            MOV   #A,R4
            MOV   #BUFF,RØ
            MOV   #BUFF1,R1
A:          MOV   (RØ)+,(R1)+
             .
             .
             .
            JMP   Ø(R4)
```

In questo modo l'istruzione JMP trasferisce il controllo all'istruzione con etichetta A, se il valore della seconda voce dell'istruzione JMP viene modificato in x durante l'esecuzione del programma, l'esecuzione dell'istruzione JMP trasferirà il controllo non all'istruzione con etichetta A ma ad una istruzione che dista x da quella con etichetta A.

6.2.2. Le istruzioni di salto condizionato

Con questo tipo di istruzioni viene controllato lo stato dei bits Ø-3 dalla Program Status word, ed in base a questo viene eseguito o meno il salto.

► BNE branch if not equal (to zero)

codice ottale	ØØ1ØØØ + offset

Controlla lo stato del bit Z e causa il salto se è Z = Ø. Si può quindi controllare se il risultato dell'operazione precedente è diverso da zero, e in tal caso eseguire il salto.

Esempio

```
CMP  A,B
BNE  C
```

Se A ≠ B il controllo del programma è trasferito all'indirizzo C.

```
ADD  A,B
BNE  C
```

Salta a C se A + B ≠ Ø.

► BEQ branch if equal (to zero)

codice ottale ØØ14ØØ + offset

E' l'operazione complementare della precedente: il salto viene eseguito se Z = 1, ossia se il risultato dell'operazione precedente è Ø.

Esempio

```
CMP  A,B
BEQ  C
```

Salta all'indirizzo C se A = B

```
ADD  A,B
BEQ  C
```

Salta a C se A + B = Ø.

► BPL branch if plus

codice ottale 1ØØØØØ+offset

Controlla lo stato del bit N e causa un salto se N = Ø, ossia se il risultato dell'operazione precedente è positivo.

Esempio

```
        TST  RØ
        BPL  AD
         :
AD :
```

L'istruzione TST mette a 1 il bit N se RØ è negativo, altrimenti N è posto uguale a Ø; quindi se il contenuto di RØ è positivo il controllo passa all'indirizzo AD.

► BMI   branch if minus

codice ottale   1ØØ4ØØ + offset

Controlla lo stato del bit N e causa un salto se N = 1.

Esempio

```
        TST  RØ
        BMI  AD
AD :
```

In questo caso il controllo del programma passa all'indirizzo AD se il contenuto di RØ è negativo.

► BVC   branch if overflow is clear

codice ottale   1Ø2ØØØ + offset

Controlla lo stato del bit V (bit di overflow) e causa un salto se il bit V è Ø.

► BVS branch if overflow is set

codice ottale 1Ø24ØØ + offset

Controlla lo stato del bit V (bit di overflow) e causa un salto se V = 1.
Questa istruzione è usata per sapere se l'operazione precedente ha causato un overflow aritmetico.

Esempio

Le due sequenze

```
A :  ADD  R1,R1           A :  ADD  R1,R1
     BVC  A                    BVS  AD
AD :                            BR  A
                         AD :
```

sono equivalenti. Infatti in entrambe si passa il controllo all'indirizzo AD quando nell'operazione di addizione si raggiunge l'overflow aritmetico.

► BCC branch if carry is clear

codice ottale 1Ø3ØØØ + offset

Controlla lo stato del bit C e causa un salto se C ≠ Ø.

► BCS branch if carry is set

codice ottale 1Ø34ØØ + offset.

Controlla lo stato del bit C e causa un salto se C = 1. Questa istruzione è usata per controllare se c'è stato un riporto nel risultato di una operazione precedente.

Esempio

Consideriamo un problema per calcolare e stampare una tavola di potenze del due. Cioè per calcolare 2^n con n = ∅,1,2,... e stampare n e 2^n; il calcolo dovrà terminare appena si verifica un overflow. Invece di ottenere le potenze del 2 mediante successive moltiplicazioni per 2, per il nostro programma usiamo le addizioni successive: 1 + 1 = 2, 2 + 2 = 4, 4 + 4 = 8..... Le potenze vengono trasferite nel buffer di 15_{10} parole all'indirizzo POT, mentre gli esponenti successivi del 2 vengono trasferiti nel buffer EXP.

```
INIZIO:
        MOV R0,-(SP)            ; SI SALVA IL CONTENUTO DEI
        MOV R1,-(SP)            ; REGISTRI 0,1,2,3 SULLO STACK
        MOV R2,-(SP)
        MOV R3,-(SP)
        CLR R0                  ; SI AZZERANO I REGISTRI
        CLR R1
        CLR R2
        CLR R3
        TST R0                  ; SI METTONO A 0 I BITS N,V,C
        INC R1
        MOV #EXP,R2             ; SI PONE IN R2 L' INDIRIZZO
                                ; DEL BUFFER CHE CONTIENE
                                ; GLI ESPONENTI
        MOV #POT,R3             ; SI PONE IN R3 L' INDIRIZZO
                                ; DEL BUFFER CHE CONTIENE
                                ; LE POTENZE DEL 2
LOOP:
        BVS EXIT                ; SI CONTROLLA SE SI E' RAGGIUNTO
                                ; L'OVERFLOW
        MOV R0,(R2)+            ; R0 CONTIENE N
        MOV R1,(R3)+            ; R1 CONTIENE LA POTENZA N-ESIMA
        INC R0                  ; SI INCREMENTA N
        ADD R1,R1               ; SI OTTIENE LA POTENZA SUCCESSIVA
        BR LOOP
EXIT:
        MOV (SP)+,R3
        MOV (SP)+,R2            ; SI RIPRISTINA IL CONTENUTO
        MOV (SP)+,R1            ; DEI REGISTRI
        MOV (SP)+,R0
        .EXIT
; DEFINIZIONE DEI FILES DI I/O
;
;
;
POT:    .BLKW 17
EXP:    .BLKW 17
        .END INIZIO
```

Come si può vedere da questo esempio in ogni programma c'è un punto di ingresso (in questo caso INIZIO) che indica il punto di inizio dell'esecuzione del programma stesso, e la frase .EXIT (cfr.§32) che indica la fine dell'esecuzione. La fine fisica del programma è indicata dalla frase .END (cfr.§17.12).
Per creare un buffer si usa la direttiva .BLKW (cfr.§17.11).
Inoltre, poiché l'elaboratore non può sapere a priori se una voce in memoria è una effettiva istruzione in linguaggio macchina o un dato dovuto da una pseudo istruzione o direttiva, il programmatore deve situare le direttive in modo opportuno, ossia all'inizio o alla fine del programma, dopo la frase .EXIT
Le stringhe precedute da ";" sono commenti.

6.2.3. <u>Le istruzioni di salto condizionato con segno</u>

Queste istruzioni sono usate per controllare, con particolari combinazioni dei bits condition codes, il risultato di istruzioni in cui gli operandi sono considerati valori con il segno.
Il significato dei confronti con il segno differisce da quello senza segno, nel senso che la sequenza dei valori nell'aritmetica a 16 bits con complemento a due è la seguente:

```
Ø77777   più grande numero positivo
Ø77776
  .
  .
  .
ØØØØØ1
ØØØØØØ
177777
177776
  .
  .
  .
1ØØØØ1
1ØØØØØ   più piccolo numero negativo
```

mentre nell'aritmetica a 16 bits senza segno la sequenza è:

177777 numero più grande
:
ØØØØØ2
ØØØØØ1
ØØØØØØ numero più piccolo

► BGE branch if greater than or equal (to zero)

codice ottale ØØ2ØØØ + offset

Causa un salto se N e V sono entrambi Ø o entrambi 1.(N ∀ V= Ø)(1).
BGE è l'operazione complementare di BLT. Così causerà un salto quando segue un'operazione che addiziona due numeri positivi; si avrà il salto anche se il risultato è zero.

► BLT branch if less than (zero)

codice ottale ØØ24ØØ + offset

Causa un salto se l' or esclusivo dei bits N e V è 1 (N ∀ V = 1).
Così BLT causerà sempre un salto se segue una operazione di addizione di due numeri negativi, anche se avviene l'overflow.
In particolare BLT causerà sempre un salto se segue una istruzione CMP con operando sorgente negativo e destinazione positiva (anche se avviene l'overflow). Inoltre BLT non causerà mai un salto quando segue una istruzione CMP con operando sorgente positivo e destinazione negativa, oppure quando il risultato dell'operazione precedente è zero.

---

(1) - L'operatore ∀ è l'operatore OR esclusivo.

► BGT branch if greater than (zero)

codice ottale 003000 + offset

E' un'operazione simile a BGE, eccetto che BGT non causerà un salto quando il risultato è zero.

► BLE branch if less than or equal (to zero)

codice ottale 003400 + offset

Operazione simile a BLT, ma causerà un salto anche se il risultato dell'operazione precedente è zero.

6.2.4. Le istruzioni di salto condizionato senza segno

Queste istruzioni procurano un mezzo per controllare il risultato di operazioni di confronto nelle quali gli operandi sono considerati valori senza segno.

► BHI branch if higher

codice ottale 101000 + offset

Il salto avviene quando l'operazione precedente non causa né un riporto né un risultato nullo.
Questo avverrà nelle operazioni di confronto (CMP) quando l'operando sorgente ha un valore senza segno maggiore di quello della destinazione.

Esempio

Se MUNO ha valore 177777 e DUE ha valore ØØØØØ2, nella sequenza:

```
CMP   #MUNO,#DUE
BGE   AD
```

non avviene il salto all'indirizzo AD, mentre nella sequenza

```
CMP   #MUNO,#DUE
BHI   AD
```

avviene il salto all'indirizzo AD perché le quantità sono considerate senza segno.

► BLOS   branch if lower or same

codice ottale   1Ø14ØØ + offset

Il salto avviene se l'operazione precedente causa un riporto oppure un risultato nullo. BLOS è l'operazione complementare di BHI. Il salto avverrà nelle operazioni di confronto quando l'operando sorgente ha un valore senza segno uguale o minore della destinazione.

► BHIS   branch if higher or same

codice ottale   1Ø3ØØØ + offset

E' la stessa istruzione di BCC; questo nome simbolico è incluso solo per convenienza.

► BLO   branch if lower

codice ottale   1Ø34ØØ + offset

E' la stessa istruzione di BCS; questo nome simbolico è incluso solo per convenienza.

6.2.5. L'istruzione di salto controllato

► SOB subtract one and branch (if ≠ ∅)

codice ottale ∅77R∅∅ + offset

La sintassi dell'istruzione è:

SOB R,A

Il registro R è decrementato di 1; se non è uguale a zero, l'offset , moltiplicato per due, è tolto dal valore attuale del PC, per cui si fa un salto all'indirizzo A. Questa istruzione procura un metodo veloce ed efficiente per il controllo di un ciclo. Non può però essere usata per fare salti in avanti.

Esempio

Consideriamo ora un segmento di programma per trasferire i dati contenuti in un buffer in un altro buffer; sia BUFF1 l'indirizzo del buffer di dieci voci che contiene i dati e sia BUFF2 il buffer dove i dati vengono trasferiti

```
INIZIO:
        MOV #12,R0          ; SI PONE IN R0 IL NUMERO
                            ; ( OTTALE) DEI DATI DA TRASFERIRE
        MOV #BUFF1,R1       ; SI PONE IN R1 L' INDIRIZZO
                            ; DI BUFF1
        MOV #BUFF2,R2       ; SI PONE IN R2 L' INDIRIZZO
                            ; DI BUFF2
TRAF:   MOV (R1)+,(R2)+     ; TRASFERIMENTO DATI
        SOB R0,TRAF         ; SI CONTROLLA SE TUTTI I DATI
                            ; SONO STATI TRASFERITI
        .
        .
        .
        .
        .
        .EXIT
BUFF1:  .BLKW 10.
BUFF2:  .BLKW 10.
        .END INIZIO
```

6.3. Le istruzioni di scorrimento (shift)

Le istruzioni di scorrimento (shift) servono per spostare, ruotare, a destra o a sinistra il contenuto dell'indirizzo destinazione. Nelle istruzioni di scorrimento i bits spostati fuori sono perduti; le istruzioni di rotazione operano sulla destinazione e il bit C come se fossero un buffer circolare di 17 bits. Queste istruzioni facilitano il controllo sequenziale dei bits e una loro dettagliata manipolazione.
La definizione delle istruzioni è la seguente:

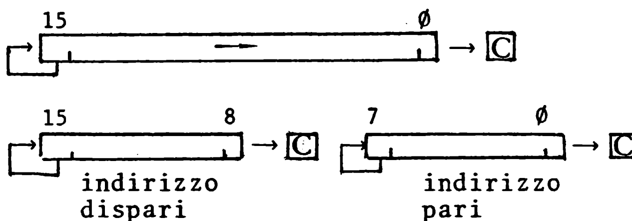
► ASR

aritmetic shift right

► ASRB

codice ottale *Ø62DD

I bits della voce all'indirizzo destinazione sono spostati di un posto verso destra. Il bit 15 è replicato; nel bit C viene messo il bit Ø della destinazione. Quando lo scorrimento è fatto su un byte, se è ad un indirizzo pari il bit C viene caricato con il bit Ø e il bit 7 è replicato; se il byte è ad un indirizzo dispari, nel bit C viene messo il bit 8, e viene replicato il bit 15.



N = 1 se il bit di ordine più alto è 1 (risultato < Ø); altrimenti è azzerato

Z = 1 se il risultato è Ø; altrimenti è azzerato

V : viene caricato con il valore di N ∀ C; il valore di C considerato è quello dopo il completamento dell'operazione

C : viene caricato con il bit di ordine più basso.

L'istruzione ASR divide il contenuto della destinazione per due, tenendo conto del segno.

Esempi

ASR RØ

prima	dopo
(RØ) = 1ØØØ111ØØØ11ØØ11	(RØ) = 11ØØØ111ØØØ11ØØ1
N Z V C	N Z V C
Ø Ø Ø Ø	1 Ø Ø 1

ASR R1

prima	dopo
(R1) = ØØØØØØØØØ1ØØØØØØ	(RØ) = ØØØØØØØØØØ1ØØØØØ
N Z V C	N Z V C
Ø Ø Ø Ø	Ø Ø Ø Ø

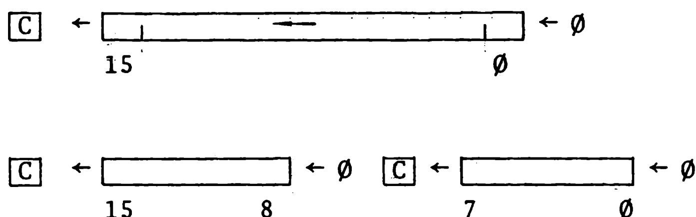
► ASL

aritmetic shift left

► ASLB

codice ottale *Ø63DD

I bits della destinazione sono spostati di un posto verso sinistra. Il bit Ø è caricato con uno zero; il bit C è caricato con il bit più significativo della destinazione



N = 1 se il bit di ordine più alto del risultato è 1 (risultato < Ø); altrimenti è azzerato

Z = 1 se il risultato è zero; altrimenti è azzerato

V : è caricato con il valore di N ∀ C

C : è caricato con il bit di ordine più alto.

L'istruzione ASL moltiplica il contenuto della destinazione per due, tenendo conto del segno e indicando l'overflow.

Esempio

ASL RØ

prima	dopo
(RØ) = ØØØØØØØØØ1ØØØØØØ	(RØ) = ØØØØØØØØ1ØØØØØØØ
N Z V C	N Z V C
1 Ø Ø Ø	Ø Ø Ø Ø

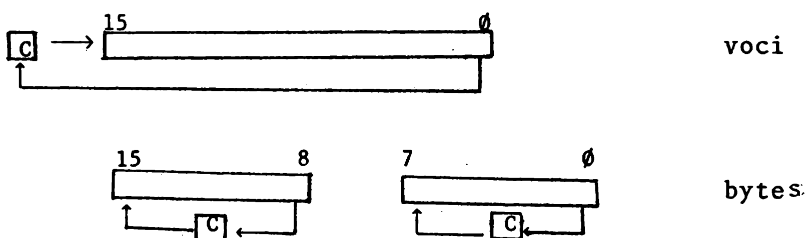
► ROR

rotate right

► RORB

codice ottale *Ø6ØDD

Tutti i bits della destinazione sono ruotati di un posto verso destra; il bit Ø è posto nel bit C e il contenuto precedente del bit C è posto nel bit 15 della destinazione.



N = 1 se il risultato è < Ø; altrimenti è azzerato

Z = 1 se il risultato è Ø; altrimenti è azzerato

V = N ∀ C

C : è caricato con il bit di ordine più basso della destinazione.

Esempio

ROR R1

prima	dopo
(R1) = Ø1Ø1Ø1Ø1Ø1Ø1Ø1ØØ	(R1) = 1Ø1Ø1Ø1Ø1Ø1Ø1Ø1Ø
N Z V C	N Z V C
Ø Ø 1 1	1 Ø 1 Ø

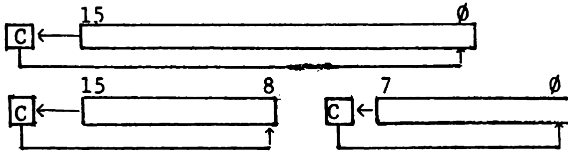
► ROL

rotate left

► ROLB

codice ottale *Ø61DD

I bits della destinazione sono ruotati di un posto verso sinistra. Il bit di ordine più alto è caricato nel bit C, e il valore precedente del bit C è messo nel bit di ordine più basso della destinazione. L'effetto sugli altri condition codes è lo stesso che per l'istruzione ROR.



Esempio

ROL RØ

prima	dopo
(RØ) = Ø1Ø1Ø1Ø1Ø1Ø1Ø1ØØ	(RØ) = 1Ø1Ø1Ø1Ø1Ø1Ø1ØØ1
N Z V C	N Z V C
Ø Ø 1 1	1 Ø 1 Ø

► SWAB swap bytes

codice ottale ØØØ3DD

Scambia i due bytes della parola all'indirizzo destinazione.

N = 1 se il bit 7 del risultato è = 1; altrimenti è azzerato

Z = 1 se il byte di destra del risultato è zero; altrimenti è azzerato

V : azzerato

C : azzerato.

Esempio

SWAB R1

prima	dopo
(R1) = Ø77777	(R1) = 177577
N Z V C	N Z V C
1 1 1 1	Ø Ø Ø Ø

6.3.1. Le istruzioni di shift multiplo

Queste istruzioni sono possibili solo con l'opzione EIS (Extended Instruction Set); la sintassi di tali istruzioni è:

OPR src,R

dove R è un registro generale, e "src" è l'operando sorgente.

► ASH shift arithmetically

codice ottale Ø72RSS

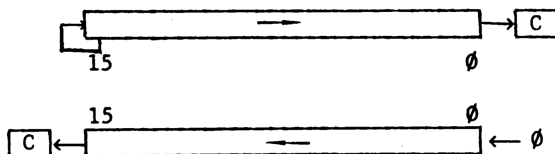
I sei bits di ordine più basso dell'operando sorgente sono considerati come contatore ("shift count") ossia il loro valore indica il numero di posizioni da spostare; il contenuto del registro è quindi spostato a destra o a sinistra del numero di volte specificato dallo shift count. Questo numero varia da -32 a +31; se è negativo indica uno shift a destra, se è positivo uno shift a sinistra.

N = 1 se il risultato è < Ø; altrimenti è azzerato

Z = 1 se il risultato è zero; altrimenti è azzerato

V = 1 se il segno del registro cambia durante lo shift; altrimenti è azzerato

C : è caricato con l'ultimo bit spostato fuori dal registro.



Negli scorrimenti verso destra il bit 15 è replicato; negli scorrimenti verso sinistra le posizioni di ordine più basso sono riempite con zeri.

► ASHC arithmetic shift combined

codice ottale Ø73RSS

Per illustrare questa istruzione è necessario distinguere due casi, ossia quando il registro usato ha numero pari, oppure dispari. Viene infatti fatto un "OR" logico del numero del registro con 1 e considerato il registro che ha come numero il risultato di questa operazione; pertanto se il numero del registro è pari si ottiene il registro successivo (R+1), altrimenti si ottiene R stesso. Consideriamo infatti, per esempio, $R = 4_{1Ø}$; allora Rv1 = 1ØØv1 = 1Ø1 = $5_{1Ø}$ mentre se $R = 3_{1Ø}$, allora Rv1 = Ø11v1 = Ø11 = $3_{1Ø}$.
Nel primo caso i due registri sono considerati come una parola a 32 bits, R+1(bits Ø-15) e R (bits 16-31) e sono spostati a destra o a sinistra del numero di volte specificato dallo "shift count". Anche in questa istruzione lo "shift count" è dato dai 6 bits di ordine più basso dell'operando sorgente, per cui varia da -32 a +31. Uno shift count negativo corrisponde ad uno scorrimento verso destra ed uno positivo ad uno scorrimento verso sinistra.

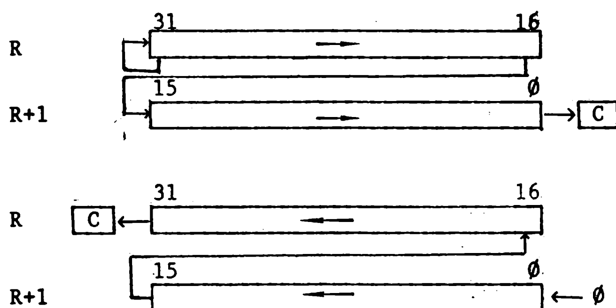
Quando il registro scelto è dispari lo scorrimento a destra corrisponde ad una rotazione (fino ad un massimo di 16 bits).

N = 1 se il risultato è < Ø; altrimenti è azzerato

Z = 1 se il risultato = Ø; altrimenti è azzerato

V = 1 se il segno cambia durante lo scorrimento; altrimenti è azzerato

C : caricato con l'ultimo bit spostato.



Esempio

Supponiamo di avere tre quantità definite dai loro indirizzi simbolici A,B,C e di voler rimpiazzare B con 4A+B e C con 3A/4-C. Questo può essere fatto con il seguente segmento di programma.

```
PRIMO:
        MOV  A,R0
        ASL  R0                    ; MOLTIPLICA IL CONTENUTO DI R0 PER 2
        ASL  R0                    ; R0 CONTIENE 4A
        ADD  R0,B                  ; AGGIUNGE 4A A B
        MOV  A,R0
        ASR  R0                    ; DIVIDE PER 2 R0
        MOV  R0,A                  ; SOSTITUISCE A CON A/2
        ASR  R0                    ; R0 CONTIENE A/4
        ADD  A,R0                  ; R0 CONTIENE 3A/4
        SUB  C,R0                  ; SOTTRAE C DA 3A/4
        MOV  R0,C
```

6.4. Moltiplicazione e divisione

Anche queste istruzioni sono possibili solo con l'opzione EIS. Per moltiplicare due quantità in memoria, una di esse deve essere contenuta in un registro, che compare come operando destinazione nell'istruzione:

►MUL multiply

codice ottale ∅7∅RSS

I contenuti del registro destinazione e dell'operando sorgente sono moltiplicati e il prodotto è memorizzato nel registro destinazione e nel registro successivo (se R è un registro pari). Se R è dispari è memorizzata solo la parte bassa del prodotto.

N = 1 se il prodotto è < ∅; altrimenti è azzerato

Z = 1 se il prodotto è ∅; altrimenti è azzerato

V : azzerato

C = 1 se il risultato è minore di -2^{-15} o maggiore o uguale a $2^{15}-1$.

Esempi

MUL #2,R∅

Il contenuto del registro ∅ è moltiplicato per 2 e il risultato è memorizzato in R∅ e R1. Pertanto se il prodotto può essere contenuto in una voce a 16 bits, R∅ conterrà tutti zero, mentre R1 conterrà il prodotto.

MUL R2,R∅

I contenuti di R∅ e R2 sono moltiplicati e il prodotto è memorizzato in R∅ e R1.

MUL RØ,R1

I contenuti di RØ e R1 sono moltiplicati e il prodotto è memorizzato solo in R1, mentre RØ contiene il valore originale.

Esempio.

Per calcolare l'area e il perimetro di un rettangolo ABCD si può procedere nel modo seguente:

```
PRIMO:
        MOV AB,R0
        ADD CD,R0
        ADD R0,R0
        MOV R0,PER                  ; R0 CONTIENE IL PERIMETRO
        MOV AB,R0
        MUL CD,R0
        ASHC #16.,R0
        MOV R0,AREA                 ; R0 CONTIENE L' AREA
        .
        .
        .
        .
        .
        .EXIT
; DEFINIZIONE DEI FILES DI I/O
AB:     10.
CD:     20.
PER:    .BLKW 1
AREA:   .BLKW 1
        .END PRIMO
```

► DIV divide

codice ottale Ø71RSS

Il registro usato (operando destinazione) deve essere pari. Il dividendo (a 32 bits),che deve trovarsi nel registro e nel successivo, è diviso per l'operando sorgente. Il quoziente resta

nel registro indicato nell'istruzione e il resto nel registro successivo.

N = 1 se il quoziente è < Ø; altrimenti è azzerato

Z = 1 se il quoziente è Ø; altrimenti è azzerato

V = 1 se l'operando sorgente è Ø oppure se il valore assoluto del registro è più grande del valore assoluto dell'operando sorgente

C = 1 se si tenta di dividere per Ø; altrimenti è azzerato.

Es.

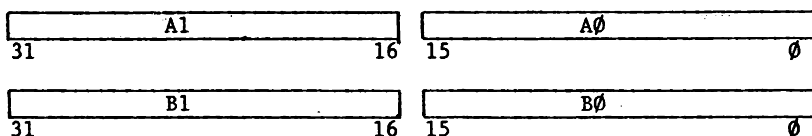
```
CLR  RØ
MOV  #2ØØ1,R1
DIV  #2,RØ
```

Si divide 2ØØ1 per 2; RØ conterrà il quoziente mentre R1 conterrà il resto.

prima	dopo	
(RO) = ØØØØØØ	(RO) = ØØØ1ØØØ	quoziente
(R1) = ØØ2ØØ1	(R1) = ØØØØØ1	resto

6.5. Aritmetica in doppia precisione

Talvolta è necessario operare su quantità in doppia precisione. Questo può essere fatto con particolari istruzioni. Le parole in doppia precisione possono essere considerate come segue:



Per sommare due dati in doppia precisione si può procedere nel modo seguente:

► ADC

add carry

► ADCB

codice ottale *Ø55DD

Aggiunge il contenuto del bit C alla destinazione. Questo permette che il riporto dall'addizione delle parole di ordine più basso degli operandi venga aggiunto al risultato dell'addizione delle parole di ordine più alto.

N = 1 se il risultato è < Ø; altrimenti è azzerato

Z = 1 se il risultato è Ø; altrimenti è azzerato

V = 1 se il contenuto della destinazione era Ø77777 e C era 1; altrimenti è azzerato

C = 1 se il contenuto della destinazione era 177777 e C era 1; altrimenti è azzerato.

Esempio

Facendo riferimento alla figura precedente, l'addizione di -1 e -1 potrebbe essere eseguita come segue:

prima dopo

(AØ) = 177777 (A1) = 177777 (BØ) = 177777 (B1) = 1777777

```
ADD  AØ,BØ
ADC  B1
ADD  A1,B1
```

dopo l'addizione di AØ e BØ, il bit C viene posto a 1; l'istruzione ADC aggiunce C a B1, per cui (B1)=Ø; poi viene aggiunto A1

a B1; il risultato (a 32 bits) è quindi 3777777777 , cioè -2.

► SBC

subtract carry

► SBCB

codice ottale *Ø56DD

Sottrae il contenuto del bit C dalla destinazione. Questo fa si che il riporto dalla sottrazione delle parole di ordine più basso sia sottratto dal risultato della parte di ordine più alto.

N = 1 se il risultato è < Ø; altrimenti è azzerato

Z = 1 se il risultato è Ø; altrimenti è azzerato

V = 1 se il contenuto della destinazione era 1ØØØØØ; altrimenti è azzerato

C = Ø se il contenuto della destinazione era Ø e C era 1; altrimenti è 1.

► SXT sign extend

codice ottale ØØ67DD

Se il bit N è uguale a 1, nell'operando destinazione viene posto -1; se il bit N è Ø, allora viene posto Ø nell'operando destinazione. Questa istruzione è particolarmente utile nell'aritmetica in precisione multipla perché permette l'estensione del segno nelle parole multiple.

N = inalterato

Z = 1 se N = Ø

V : azzerato

C : inalterato.

Esempio

SXT A

prima		dopo	
	N Z V C		N Z V C
(A) = Ø12345	1 Ø Ø Ø	(A) = 177777	1 Ø Ø Ø

6.6. Le istruzioni logiche

Queste istruzioni sono particolarmente utili per la manipolazione dei bits.

► BIT

bit test

► BITB

codice ottale *3SSDD

Esegue un "and" logico dei due operandi e modifica i condition codes in base al risultato. I due operandi restano inalterati. Serve quindi per controllare se ad un bit 1 dell'operando destinazione corrisponde un bit 1 dell'operando sorgente; analogamente per un bit Ø.

N = 1 se il bit di ordine più alto nel risultato è 1; altrimenti è azzerato

Z = 1 se il risultato è Ø; altrimenti è azzerato

V : azzerato

C : inalterato.

Esempio

BIT #3Ø,R3

Poiché $3Ø_8$ = Ø ØØØ ØØØ ØØØ Ø11 ØØØ, controlla i bits 3 e 4 del

registro 3.

► BIC

bit clear

► BICB

codice ottale *4SSDD

Azzera ciascun bit nella destinazione che corrisponde ad un bit uguale a 1 nell'operando sorgente. Il contenuto originale della destinazione è perduto, mentre quello dell'operando sorgente è inalterato.

N = 1 se il bit di ordine più alto del risultato è 1; altrimenti è azzerato

Z = 1 se il risultato è zero

V : azzerato

C : inalterato

Esempio

BIC RØ,R1

prima

(RØ) = Ø ØØØ 1ØØ ØØ1 1Ø1 Ø1Ø

(R1) = Ø ØØØ 1Ø1 ØØØ 11Ø Ø11

dopo

(RØ) = Ø ØØØ 1ØØ ØØ1 1Ø1 Ø1Ø

(R1) = Ø ØØØ ØØ1 ØØØ Ø1Ø ØØ1

► BIS

bit set

► BISB

codice ottale *5SSDD

Esegue un operazione di "OR" inclusivo fra l'operando sorgente e quello destinazione; il risultato resta nell'operando destinazione. Pertanto viene messo a 1 ciascun bit dell'operando destinazione che corrisponde ad un 1 nell'operando sorgente. Il contenuto originale della destinazione è perso (1).

Esempi

BIS R1,R2

prima

(R1) = Ø 1Ø1 11Ø 111 ØØ1 Ø1Ø

(R2) = Ø ØØØ ØØØ ØØØ ØØØ ØØØ

dopo

(R1) = Ø 1Ø1 11Ø 111 ØØ1 Ø1Ø

(R2) = Ø 1Ø1 11Ø 111 ØØ1 Ø1Ø

BIS RØ,R2

prima

(RØ) = Ø ØØ1 1Ø1 1ØØ 11Ø Ø1Ø

(R2) = Ø 1ØØ 1ØØ Ø11 Ø11 ØØØ

---

(1) N = 1 se il bit di ordine più alto del risultato è 1; altrimenti è azzerato

Z = 1 se il risultato è zero; altrimenti è azzerato

V = Ø

C : inalterato.

dopo

(RØ) = Ø ØØ1 1Ø1 1ØØ 11Ø Ø1Ø

(R2) = Ø 1Ø1 1Ø1 111 111 Ø1Ø

► XOR exclusive OR

codice ottale ØZ4RDD

In questa istruzione l'operando sorgente è un registro generale; esegue l'"OR" esclusivo fra il contenuto del registro e l'operando destinazione. Il risultato di tale operazione sostituisce il contenuto originale dell'indirizzo destinazione.
(1)

Esempio

XOR RØ,R1

prima

(RØ) = Ø Ø1Ø 11Ø 111 Ø1Ø 1Ø1

(R2) = Ø ØØ1 Ø1Ø Ø11 ØØ1 Ø1Ø

dopo

(RØ) = Ø Ø1Ø 11Ø 111 Ø1Ø 1Ø1

(R1) = Ø Ø11 1ØØ 1ØØ Ø11 111

---

(1) N = 1 se il risultato è < Ø ; altrimenti è azzerato

Z = 1 se il risultato è Ø; altrimenti è azzerato

V = Ø

C : inalterato.

Esempio

L'esempio seguente illustra come, mediante le operazioni logiche, sia possibile modificare a programma il significato delle istruzioni.

```
INIZIO:
        .
        .
        .
        CLR R0
        CLR R1
        CLR R2
        INC R2
        MOV #A,R4
A:      INC R1
        MOV A,BUFF+2
B:      MOV R1,BUFF
        MOV B,BUFF+4
        INC R0
        CMP #2,R0
        BEQ FINE
        INC A
        BIS #200,B
        BIC #100,B
        BR A
FINE:
        .
        .
        .
        .EXIT
BUFF:   .BLKW 3
        .
        .
        .
        .END INIZIO
```

L'istruzione INC A, incrementa il contenuto dell'indirizzo A; ma tale contenuto è l'istruzione INC, che pertanto, (in ottale) diventa:

ØØ 52Ø2

che equivale a

INC R2

con l'istruzione

BIS #2ØØ,B

viene messo a 1 il bit 7 di B, mentre con la successiva istruzione

BIC #1ØØ,B

si azzera il bit 6 di B. Pertanto il contenuto di B diventa (in ottale)

010267

che corrisponde a:

MOV R2,BUFF.

Pertanto, in seguito all'istruzione

BR A

il contenuto del registro 2 verrà incrementato e poi trasferito in BUFF.

7. I SOTTOPROGRAMMI

Le informazioni necessarie nei collegamenti con i sottoprogrammi possono essere così riassunte:

a) posizione di memoria a cui si trova la prima istruzione eseguibile del sottoprogramma

b) indirizzo a cui ritornare nel programma principale dopo che il sottoprogramma ha eseguito le sue operazioni

c) campi sui quali il sottoprogramma deve operare.

I sottoprogrammi ("subroutines") sono chiamati con l'istruzione

► JSR	jump to subroutine
codice ottale	ØØ4RDD

che ha il seguente formato:

JSR R,SUBR

dove R è un registro generale (detto *LINKAGE POINTER*) e SUBR è il punto d'ingresso della subroutine.
Il punto di ingresso del sottoprogramma viene messo in una posizione temporanea; il contenuto del registro usato nella istruzione è salvato nello stack e poi sostituito con il valore attuale del PC, che adesso punta alla istruzione successiva alla istruzione JSR. Pertanto il registro contiene l'indirizzo di ritorno dal sottoprogramma. Nel PC viene poi trasferito il punto di ingresso del sottoprogramma, realizzando così il collegamento.

Esempio

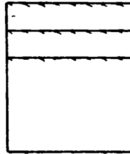
JSR R5, SUBR

prima

(R5) = ØØØØ1Ø

(R7) = 15736Ø

SP →

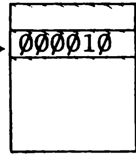


dopo

(R5) = 157362

(R7) = SUBR

SP→



Poiché il contenuto del registro usato per il collegamento è automaticamente salvato nello stack, i sottoprogrammi possono essere inseriti uno dentro l'altro e richiamati usando lo stesso registro.
Le posizioni di memoria che seguono l'istruzione JSR, e di cui, quindi, il registro usato per il collegamento contiene l'indirizzo, possono contenere i parametri o gli indirizzi dei parametri su cui il sottoprogramma deve operare. Il primo parametro può quindi essere reperito usando i modi di indirizzamento (R5), (R5)+, x(R5) (se R5 è il registro usato) oppure @(R5)+ se i parametri sono passati per indirizzo. Usando l'autoincremento il contenuto del registro è automaticamente aggiornato per puntare al dato successivo.

Esempio.

Supponiamo che il sottoprogramma SUBR debba operare sui valori 1ØØ e 15Ø, che si trovano rispettivamente agli indirizzi A1ØØ e A15Ø, trasferendoli nei registri 1 e 2.
Se la sequenza di chiamata è la seguente:

```
JSR  R5, SUBR
1ØØ
15Ø
 .
 .
 .
```

la trasmissione dei parametri si ottiene nel modo seguente:

```
SUBR : MOV  (R5)+,R1
       MOV  (R5)+,R2
```

Se invece la sequenza di chiamata è la seguente:

```
JSR  R5,SUBR
A1ØØ
A15Ø
```

la trasmissione dei parametri si ottiene con l'autoincremento differito del registro 5 :

```
SUBR : MOV  @(R5)+,R1
       MOV  @(R5)+,R2
```

Se non deve essere trasmesso alcun parametro, oppure i parametri sono in un registro generale o nello stack la chiamata del sottoprogramma può essere fatta con l'istruzione

```
JSR  PC,SUBR
```

cioè usando come registro di collegamento il "program counter". Gli unici registri modificati con questo tipo di chiamata sono R7 e SP.
Il ritorno al programma chiamante si ottiene con l'istruzione

► RTS   return from subroutine

codice ottale   ØØØ2ØR

che ha il formato:

```
RTS  R
```

dove R è il registro usato nella chiamata del sottoprogramma. Il valore attuale del registro è trasferito nel PC, e il valore originale del registro, prima della chiamata del sottoprogramma è ripristinato togliendolo dallo stack.
Un sottoprogramma, SUBR, chiamato con

```
JSR  R5,SUBR
```

può reperire i parametri utilizzando il registro 5 e infine termina con l'istruzione

```
RTS  R5.
```

La chiamata di un sottoprogramma si può realizzare anche con una istruzione JMP; in questo caso, però, non è usato nessun registro di collegamento e non c'è modo di ritornare al programma chiamante.
I parametri di un sottoprogramma possono essere trasmessi attraverso lo stack. In questo caso è necessario toglierli dallo stack quando il sottoprogramma termina. Ciò si può fare a programma, tenendo nella prima voce dello stack il numero dei parametri, oppure usando l'istruzione:

MARK	mark	
codice ottale	ØØ64NN	(NN = numero parametri)

che ha il formato:

```
MARK  N.
```

Per illustrare il funzionamento di questa istruzione consideriamo il seguente esempio:

```
MOV  R5,-(SP)
MOV  A1,-(SP)
MOV  A2,-(SP)
      :
MOV  AN,-(SP)
MOV  #MARKN,-(SP)
MOV  SP,R5
JSR  PC,SUBR
```

Si salva il contenuto originale del registro 5 e si pongono gli N parametri sullo stack, seguiti dall'istruzione MARK N ; si salva il valore attuale di SP (ossia l'indirizzo della voce sullo stack che contiene l'istruzione MARK N), nel registro 5; poi si salta al sottoprogramma SUBR.
A questo punto nello stack si ha la configurazione seguente:

R5
A1
A2
....
....
AN
MARK N
PC

Dopo che il sottoprogramma ha terminato le sue operazioni si ritorna al programma chiamante con l'istruzione:

```
RTS  R5
```

per cui il contenuto di R5 viene posto nel PC, risultando così nell'istruzione MARK N. Il contenuto del PC, prima della chiamata del sottoprogramma, è posto nel registro 5.

In seguito all'istruzione MARK N , SP viene modificato fino a puntare al valore originale di R5 ($SP \leftarrow SP+2 \times N$), il valore attuale del registro 5 viene posto nel PC, e infine viene ripristinato il valore originale di R5, completando così il ritorno dal sottoprogramma.

Un caso speciale dell'istruzione JSR è:

```
JSR  PC, @(SP)+
```

che scambia fra loro il contenuto del PC e l'elemento in testa allo stack. L'uso di questa istruzione consente che due sottoprogrammi ("routines") si scambino il controllo e, quando sono richiamati, riprendono le operazioni al punto dove erano state interrotte. Tali routines sono dette "co-routines".
Supponiamo per esempio che stia operando la routine #1 e che PC2 contenga il punto d'ingresso della routine #2. Ad un certo punto esegue:

```
MOV  #PC2,-(SP)
JSR  PC,@(SP)+
```

allora PC2 è tolto dallo stack e (SP) incrementato; poi SP è decrementato e PC messo sullo stack (nella posizione, quindi dove si trovava PC2); il contenuto di PC2 è messo in PC, per cui il controllo è trasferito alla routine #2. Se ad un certo punto la routine #2 esegue:

```
JSR  PC,@(SP)+
```

il controllo è di nuovo trasferito alla routine #1, al punto dove era stata interrotta.

Esempio

Nel seguente segmento di programma, si ha la chiamata di un sottoprogramma con punto d'ingresso ZERO. Il parametro è trasmesso per indirizzo.

```
        JSR R5,ZERO             ; CHIAMATA DEL SOTTOPROGRAMMA
        .WORD PAR               ; INDIRIZZO DEL PARAMETRO
        .
        .
        .
        .
        .
;
; SOTTOPROGRAMMA ZERO
;
ZERO:   MOV R1,-(SP)            ; SALVA IL CONTENUTO DEI REGISTRI
        MOV R2,-(SP)
        CLR R2                  ; AZZERA R2 CHE SERVE DA CONTATORE
        MOV (R5)+,R1            ; R1 CONTIENE L' INDIRIZZO DEL BUFFER
                                ; DEI DATI ASCII
BLK:    INC R2
        CMPB #060,(R1)          ; IL CARATTERE IN ESAME E' 0 ?
        BNE FINE                ; NO - VAI  A FINE
        CMP #5,R2               ; SI - HAI RAGGIUNTO L' ULTIMO
                                ; CARATTERE ?
        BEQ FINE                ; SI - VAI A FINE
        MOVB #40,(R1)+          ; NO - SOSTITUISCILO
                                ; CON UN CARATTERE VUOTO
        BR BLK
FINE:   MOV (SP)+,R2            ; RIPRISTINA IL CONTENUTO DEI REGISTRI
        MOV (SP)+,R1
        RTS R5
        .
        .
        .
PAR:    .BLKB 5
```

La struttura dell'Assemblatore permette di realizzare facilmente i collegamenti fra programmi in linguaggio assemblativo (assembly) e sottoprogrammi FORTRAN. Sia per esempio FTSUB il nome di un sottoprogramma FORTRAN che operi su parametri agli indirizzi PAR1, PAR2,...,PARN; la chiamata di questo sottoprogramma, da un programma assembly si realizza con la sequenza:

```
          JSR   R5,FTSUB
          BR  PAR
         .WORD  PAR1
         .WORD  PAR2
          .
          .
          .
         .WORD  PARN
PAR:
```

L'istruzione dopo la JSR deve essere una istruzione di salto che porta il controllo all' istruzione che segue, fisicamente, gli indirizzi degli n parametri. Per i collegamenti FORTRAN-MACRO cfr. Appendice B.

8. LE ISTRUZIONI IN FLOATING POINT

Queste operazioni sono possibili solo con l'opzione Floating Point.

Il formato delle istruzioni per l'aritmetica in floating point è:

OPR R

dove R è uno dei registri generali, usato come puntatore per specificare un indirizzo su uno stack. Il contenuto del registro è usato per indirizzare gli operandi e il risultato; se chiamiamo con A e B due argomenti in floating point, allora:

(R)   indirizzo della prima voce di B
(R)+2 = indirizzo della seconda voce di B
(R)+4 = indirizzo della prima voce di A
(R)+6 = indirizzo della seconda voce di A.

Dopo l'operazione in floating point il risultato è memorizzato nello stack nel modo seguente:

(R)+4 = indirizzo della prima voce del risultato
(R)+6 = indirizzo della seconda voce del risultato

dove (R) è il contenuto originale del registro usato.
Dopo l'esecuzione dell'istruzione il registro punta alla prima voce del risultato.

► FADD floating add

codice ottale Ø75ØØR

Aggiunge l'argomento A (cioè quello alle locazioni (R)+4 e (R)+6) all'argomento B (alle locazioni (R) e (R)+2) e lascia il risultato nelle locazioni dove prima si trovava A. Se il risultato in valore assoluto è $\leq 2^{-128}$, in tali locazioni viene posto Ø.

N = 1 se il risultato è < 0; altrimenti è azzerato

Z = 1 se il risultato è Ø; altrimenti è azzerato

V : azzerato
C : azzerato

► FSUB floating subtract

codice ottale Ø75Ø1R

Sottrae l'argomento B dall'argomento A e lascia il risultato nelle posizioni dove si trova A. Se la differenza è in valore assoluto $\leq 2^{-128}$ il risultato è Ø.

N = 1 se il risultato è < Ø; altrimenti è azzerato

Z = 1 se il risultato è Ø; altrimenti è azzerato

V : azzerato

C : azzerato.

► FMUL floating multiply

codice ottale Ø75Ø2R

Moltiplica l'argomento A per l'argomento B e lascia il risultato in A. Se il prodotto è in valore assoluto $\leq 2^{-128}$, il risultato è Ø

N = 1 se il risultato è < Ø; altrimenti è azzerato
Z = 1 se il risultato è Ø; altrimenti è azzerato
V : azzerato
C : azzerato

FDIV floating divide

codice ottale Ø75ØR

Divide l'argomento A per l'argomento B e lascia il risultato in A. Se il divisore (cioè B) è uguale a zero, le posizioni sono lasciate inalterate, ma vengono messi a 1 i bits V,N,C e viene interrotta l'esecuzione.

N = 1 se il risultato è < O; altrimenti è azzerato
Z = 1 se il risultato è Ø; altrimenti è azzerato
V e C azzerati

Se avviene una interruzione i condition-codes sono interpretati come segue:

V = 1 se avviene un errore (overflow, underflow, divisione per zero)
N = 1 se avviene un underflow o una divisione per zero
C = 1 se si tenta di dividere per zero
Z = Ø.

Esempio

Nel seguente segmento di programma è illustrato l'uso delle istruzioni in floating-point.
Le operazioni eseguite sono:

A + B

A - B

A × B

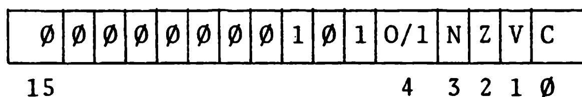
A / B.

Per eseguire queste operazioni ci serviamo di due locazioni ausiliari all'indirizzo C.

```
PRIMO:
        .
        .
        .
        MOV #B,R0
        MOV A,C
        MOV A+2,C+2
        FADD R0                  ; A+B
        MOV (R0),PIU
        MOV 2(R0),PIU+2
        MOV #B,R0
        MOV A,C
        MOV A+2,C+2
        FSUB R0                  ; A-B
        MOV (R0),MENO
        MOV 2(R0),MENO+2
        MOV #B,R0
        MOV A,C
        MOV A+2,C+2
        FMUL R0                  ; A.B
        MOV (R0),PROD
        MOV 2(R0),PROD+2
        MOV #B,R0
        MOV A,C
        MOV A+2,C+2
        FDIV R0                  ; A/B
        MOV (R0),DIV
        MOV 2(R0),DIV +2
        .
        .
        .
        .EXIT
B:      .BLKW 2
C:      .BLKW 2
A:      .BLKW 2
PIU:    .BLKW 2
MENO:   .BLKW 2
PROD:   .BLKW 2
DIV :   .BLKW 2
        .END PRIMO
```

9. LE OPERAZIONI SUI CONDITION CODES

Queste operazioni hanno il seguente formato:



e servono per azzerare o mettere a 1 i bit N,Z,V,C. Questi bits sono modificati in accordo al valore del bit 4 dell'operatore, ossia viene messo a 1 il bit specificato dal bit 0,1,2 o 3 se il bit 4 è 1; azzera i bits corrispondenti se il bit 4 è Ø.

codice simbolico		codice ottale
CLC	azzera il bit C	ØØØ241
CLV	" " " V	ØØØ242
CLZ	" " " Z	ØØØ244
CLN	" " " N	ØØØ25Ø
SEC	mette a 1 il bit C	ØØØ261
SEV	" " " " V	ØØØ262
SEZ	" " " " Z	ØØØ264
SEN	" " " " N	ØØØ27Ø
SCC	mette a 1 tutti i condition codes	ØØØ277
CCC	azzera tutti i condition codes	ØØØ257
NOP	nessuna operazione	ØØØ24Ø

Queste operazioni possono essere combinate insieme con il connettivo OR per formare istruzioni combinate.

PARTE SECONDA

LA PROGRAMMAZIONE IN MACRO-11

In questa parte sono descritte le proprietà del linguaggio assemblativo che dipendono dall'assemblatore. Alcune di queste proprietà possono, quindi, variare da una versione all'altra.

10. L'INSIEME DEI CARATTERI

L'insieme dei caratteri che si possono usare nei programmi è costituito da:

a) le lettere da A a Z (sia maiuscole che minuscole)
b) le cifre da Ø a 9
c) i caratteri . (punto)e $ che sono riservati per i simboli dei programmi di sistema
d) i seguenti caratteri speciali:

: indica la fine di una etichetta
= simbolo dell'assegnamento diretto
% simbolo di registro
indica un numero (indirizzamento immediato)
@ simbolo dell'indirizzamento differito
(,) simbolo dell'indirizzamento con registro differito
, separatore nel campo operandi
; indica l'inizio del campo commenti
<,> deliminatori di argomenti
\+ operatore aritmetico di addizione
\- " " " sottrazione
* " " " moltiplicazione
/ " " " divisione
& " logico AND
! " " OR inclusivo
" indica due caratteri ASCII
' indica un solo carattere ASCII
↑ operatore unario universale
\ indica gli argomenti numerici di una macro.

I caratteri che si possono usare come separatori sono lo spazio bianco e la virgola; per delimitatori si usano le parentesi < .. > oppure la costruzione : ↑ \.....\, dove "\ " è uno dei caratteri possibili. I caratteri < e ↑ \ possono essere considerati operatori unari che non possono essere immediatamente preceduti da un altro argomento.
Gli operatori unari sono i seguenti:

- + segno più, ossia +A è equivalente ad A
- - segno meno, ossia -A è il complemento a 2 di A
- ↑ operatore unario, il cui significato sarà illustrato nel seguito.

Gli operatori binari sono i quattro operatori aritmetici (+,-,*,/) e i due operatori logici (&,!).
Tutti gli operatori binari hanno la stessa priorità; gli argomenti possono essere raggruppati per la valutazione all'interno di una espressione racchiundendoli fra <..> ; i termini così ottenuti sono valutati per primi e le operazioni rimanenti sono eseguite da sinistra verso destra. Per esempio:

```
2 + 6 * 3 = 30   (ottale)
2 + <6 * 3>= 24  (ottale)
```

11. I SIMBOLI

Vi sono tre tipi di simboli: i simboli permanenti, i simboli definiti dall'utente, e i simboli macro. Per ognuno di questi tipi c'è la corrispondente tavola dei simboli.
La tavola dei simboli permanenti (PST) contiene tutti i simboli permanenti ed è parte del modulo eseguibile del MACRO-11 Assembler.
La tavola dei simboli definiti dall'utente (UST) e quella dei simboli macro (MST) sono costruite al momento dell'assemblaggio.

11.1. I simboli permanenti

I simboli permanenti sono i nomi simbolici delle istruzioni e le direttive assembler. Questi simboli sono una parte permanente dell'Assembler e non devono essere definiti prima del loro uso in un programma.

11.2. I simboli definiti dall'utente e i simboli MACRO

I simboli definiti dall'utente sono quelli usati come etichette o definiti con un assegnamento diretto (cfr.§13).Questi simboli sono aggiunti alla UST mano a mano che si incontrano durante il primo passo dell'assemblaggio. I simboli macro sono i simboli usati come nomi di macro e sono aggiunti alla MST durante l'assemblaggio.

I simboli definiti dall'utente e i macro simboli sono sequenze da 1 a 6 caratteri alfanumerici, caratteri $ e punti. I caratteri $ e punti sono per i simboli del software di sistema.

Il primo carattere non deve essere un numero; non sono permessi spazi bianchi all'interno di un simbolo.

Il valore di un simbolo dipende dal suo uso nel programma. Per determinare il valore di un simbolo che compare come operatore l'Assembler esamina le tre tavole nel seguente ordine:

i) tavola dei simboli macro
ii) tavola dei simboli permanenti
iii) tavola dei simboli definiti dall'utente.

Un simbolo che compare come operando è ricercato prima nella tavola dei simboli definiti dall'utente e poi in quella dei simboli permanenti.

Questo ordine di ricerca permette di definire simboli, anche macro, uguali ai simboli permanenti.

I simboli definiti dall'utente sono locali al modulo oggetto (1) a meno che non siano esplicitamente definiti esterni, cioè globali, con una direttiva .GLOBL (cfr. §17.15).
I simboli globali servono da collegamento fra i moduli oggetto; un simbolo globale definito come etichetta è generalmente detto punto di ingresso e serve per trasferire il controllo attraverso il modulo eseguibile, che è composto da un certo numero di moduli oggetto.

12. IL FORMATO DEGLI STATEMENTS

Un programma è composto da una sequenza di linee; ciascuna linea contiene una frase (statement) del linguaggio. Uno statement può contenere fino a quattro campi identificati dall'ordine di apparizione e da particolari caratteri separatori.
Il formato generale di una frase in linguaggio assemblativo è il seguente:

etichetta: operatore operando ; commenti

Il campo etichetta e il campo commenti sono facoltativi; i campi operatore e operando sono interdipendenti, ciascuno può essere omesso solo in relazione al contenuto dell'altro.
Una frase è tradotta dall'Assembler in una o più voci binarie.
Una frase del linguaggio assemblativo deve essere completata in una sola linea, non sono cioè ammesse linee di continuazione.

12.1. Il campo etichetta

Una etichetta è un simbolo definito dall'utente a cui è associato il valore corrente del contatore di locazione (location counter) cioè l'indirizzo della posizione di memoria disponibile

---

(1) Il modulo oggetto è il file, creato dall'Assembler, che contiene la traduzione in codice del programma.

in quel momento ed è inserito nella tavola dei simboli definiti dall'utente.
Se il valore dell'etichetta è rilocabile all'atto del caricamento, viene aggiunta dal programma di sistema LINK-11 la costante di rilocazione, per ottenere il vero indirizzo.
Una etichetta è quindi un indirizzo simbolico. Infatti, per esempio, se il valore assoluto della posizione è: 112_8, la frase:

```
ET1 :  MOV  #2,RØ
```

associa all'etichetta ET1 il valore 112_8 e ogni riferimento successivo a ET1 equivale ad un riferimento alla locazione 112_8.
Se il valore del location counter fosse stato rilocabile, il valore finale di ET1 sarebbe stato 112+k dove k è l'indirizzo della posizione iniziale della sezione rilocabile.
L'etichetta, quando usata, deve apparire per prima e deve sempre terminare con ":"; in una frase possono esserci più etichette, a cui è assegnato lo stesso valore. Per esempio se il valore corrente del location counter è $1ØØ_8$, nella frase:

```
ABC:  DEF:  GH1:  MOV  A,B
```

a ciascuna delle tre etichette ABC,DEF,GH1 è associato il valore $1ØØ_8$.
Un simbolo definito come etichetta non può essere ridefinito all'interno del programma.

12.2. Il campo operatore

Il campo operatore segue il campo etichetta e può contenere un nome simbolico di istruzione, una chiamata di macro, una direttiva assembler.
Se l'operatore è un nome di istruzione, questo specifica la istruzione da generare e le azioni da eseguire sull'operando

(o operandi); se è la chiamata di una macro, l'Assembler inserisce l'espansione della macro; se infine è una direttiva, specifica una certa azione da eseguire durante l'assemblaggio.
Un operatore può terminare con uno spazio bianco o con un carattere non alfanumerico; sono cioè sintatticamente corrette le frasi

```
              MOV  A,B
e             MOV@A,B
```

dove l'operatore MOV termina rispettivamente con uno spazio bianco e con un carattere non alfanumerico. (@)
Un campo operatore vuoto, è interpretato come una direttiva .WORD (cfr.§17.6).

12.3. <u>Il campo operandi</u>

Un operando è quella parte di una frase su cui agisce l'operatore, e può essere un numero, una espressione,un nome simbolico o un argomento di una macro. Se un'operazione richiede più operandi, questi sono separati da una virgola, da uno spazio bianco o dalle parentesi acute ⟨ e⟩ . Pertanto nella frase:

```
MOV  A,B
```

lo spazio bianco separa il campo operatore da quello degli operandi, mentre i due operandi A e B sono separati da virgola.

12.4. <u>Il campo commenti</u>

Il campo commenti, se è presente, deve iniziare con ";" e può contenere una sequenza qualsiasi di caratteri ASCII, esclusi caratteri nulli, ritorni di carrello, tabulatori, salto di riga e di pagina. I commenti non alterano l'esecuzione di un programma ma sono molto utili per la messa a punto e per una più facile comprensione del programma da parte degli utenti.

Poiché gli spazi bianchi non sono considerati nel processo di assemblaggio, a meno che non siano all'interno di un simbolo, di un numero o di una stringa ASCII o a meno che non siano usati come caratteri terminali del campo operatore, questi possono essere usati per migliorare la leggibilità di un programma.

13. L'ASSEGNAMENTO DIRETTO

Un altro modo per definire un simbolo è la frase di assegnamento diretto, che ha il seguente formato:

simbolo = espressione.

Se il simbolo compare per la prima volta, viene aggiunto alla tavola dei simboli e assume come valore il valore dell'espressione. Un simbolo può essere ridefinito assegnandogli un nuovo valore; l'ultimo valore assegnato sostituisce ogni altro precedente valore.
In una frase di assegnamento diretto il segno "=" separa il simbolo dall'espressione e soltanto un simbolo può essere definito; sono permessi riferimenti in avanti solo di un livello, ossia la sequenza:

```
X = Y
Y = Z
Z = 1
```

non è corretta perché il simbolo X resta indefinito.
Mentre è corretta la sequenza:

```
X = Y
Y = 1
```

Una frase di assegnamento diretto è generalmente posta nel campo operatore e può essere preceduta da una etichetta e seguita da commenti.

Le espressioni sono combinazioni di termini uniti da operatori binari, che come risultato danno un valore a 16 bits. Un termine può essere:

a) un numero. E' bene ricordare che il MACRO-11 Assembler assume che tutti i numeri siano rappresentati in base otto, a meno che non sia specificato diversamente; i numeri negativi sono preceduti dal segno meno e sono rappresentati nella forma di complemento a due; se un numero è troppo grande per essere rappresentato in 16 bits, è troncato a sinistra e questo fatto viene segnalato con un codice errore T nella lista del programma;

b) un simbolo;

c) una conversione ASCII (cfr. §17.8).

d) un termine può essere una espressione o un termine racchiuso in parentesi acute; una quantità dentro le parentesi acute è valutata prima del resto dell'espressione in cui si trova; le parentesi possono essere usate anche per alterare la valutazione da sinistra a destra di una espressione o per applicare un operatore unario ad una intera espressione. (per esempio -<A*B>).

Le espressioni sono valutate da sinistra a destra senza precedenza fra operatori; gli operatori unari precedono però quelli binari. Si possono usare operatori unari multipli e sono trattati come segue:

-+-A è equivalente a -<+<-A>>

Un termine, una espressione o simbolo esterno mancante sono interpretati come zero, mentre un operatore mancante è interpretato come +. Per esempio:

A * B 1ØØØØØ

è interpretato come:

A * B + 1ØØØØØ

In entrambi i casi viene segnalato un errore di tipo Q.

14. I SIMBOLI DI REGISTRO

I registri, come sappiamo, sono numerati da Ø a 7 e in un programma sorgente sono espressi come:

%Ø
%1
.
.
.
.
%7

dove la cifra dopo il simbolo % può essere sostituita con un termine valutabile nel primo passo dell'assemblaggio.
Ad esempio:

%2 + 2 è equivalente a %4.

E' però raccomandabile l'uso di nomi simbolici per i registri.
Un simbolo di registro è definito con un assegnamento diretto, per esempio:

```
R0 = %0
R1 = %1
R2 = %2
R3 = %3
R4 = %4
R5 = %5
R6 = %6
R7 = %7
SP = %6
PC = %7
```

I nomi simbolici usati per i registri in questo esempio, sono i nomi convenzionalmente usati; come si può vedere ai registri 6 e 7 sono dati nomi particolari, in relazione alla loro paricolare funzione.
Tutti i nomi simbolici dei registri devono essere definiti prima del loro uso nel programma. In alcuni casi particolari non è strettamente necessario l'uso del simbolo di registro; ad esempio nell'istruzione:

```
JSR 5,SUBR
```

l'uso del simbolo di registro non è necessario perché la sintassi dell'istruzione richiede che il primo operando sia un registro.

15. I SIMBOLI LOCALI

Un programma in MACRO-11 può essere costituito da blocchi in cui le etichette hanno valore locale e sono appunto dette simboli locali. Un blocco di questo tipo è delimitato in uno dei modi seguenti:

a) il rango di un blocco locale può consistere di quelle frasi (statements) che sono comprese fra due etichette simboliche normalmente costruite;

b) il rango di un blocco locale termina appena si incontra una direttiva .CSECT, .PSECT, o .ASECT;

c) il rango di un blocco locale può essere delimitato con .ENABL LSB e la prima etichetta simbolica o direttiva .CSECT, .PSECT, .ASECT che segue la direttiva .DSABL LSB.

I simboli locali hanno la forma n$ dove n è un intero decimale compreso tra 1 e 127, inclusi, e possono essere usati solo come indirizzi simbolici di parole, cioè corrispondere ad indirizzi pari. E' da tener presente che i simboli locali da 64$ a 127$ sono generati automaticamente dal macroassemblatore.
per cui è preferibile che un utente usi i simboli locali da 1$ a 63$.
L'uso di simboli locali permette di risparmiare memoria in quanto questi occupano una sola posizione in ciascun blocco contro le quattro posizioni di memoria che occupa una etichetta simbolica nella tavola dei simboli. Usando i simboli locali si riduce la possibilità di definire più volte una stessa etichetta all'interno di un programma.
La massima distanza fra un simbolo locale e la base del blocco è di 128 (decimale) parole.
Per esempio nel seguente segmento di programma:

```
        RØ = %Ø
        R1 = %1
        .CSECT BLOCK1
        CLR RØ
        INC RØ
1$:     ADD RØ,RØ
        BVC 1$
```

```
        .CSECT BLOCK2
        CLR  R1
        INC  R1
1$:     ADD  R2,R1
        CMP  R1,RØ
        BLT  1$
```

ci sono due blocchi locali che iniziano rispettivamente con .CSECT BLOCK1 e .CSECT BLOCK2, e in entrambi è possibile usare lo stesso simbolo locale 1$. E' possibile far riferimento ad un simbolo locale solo nel blocco dove è stato definito. Non c'è conflitto con etichette con lo stesso nome definite in blocchi diversi.

16. IL "LOCATION COUNTER"

Per tener conto delle posizioni di memoria esiste un dispositivo speciale chiamato "location counter"; il simbolo che lo rappresenta è il punto: ".".
All'inizio dell'assemblaggio il location counter è posto uguale a zero e generalmente ad ogni istruzione o dato vengono assegnate posizioni consecutive. Tuttavia la posizione dove i dati generati devono essere memorizzati può essere cambiata alterando il location counter con una frase di assegnamento diretto.

. = espressione

Al pari degli altri simboli il location counter può essere assoluto o rilocabile, ma non può essere esterno.
E' possibile riservare certe posizioni di memoria incrementando il location counter. Per esempio se il valore corrente del location counter è $11ØØ_8$, con la frase:

```
. = . + 200
```

si riserva uno spazio di 200_8 bytes nel programma. L'istruzione successiva è memorizzata alla locazione 1300_8.

Esempio

Nella sequenza:

```
        . = 1000

LABEL : MOV   .,ISTRUZ
```

l'etichetta LABEL ha il valore 1000_8 (rilocabile o assoluta); il contenuto della locazione 1000_8 , ossia il codice binario dell'istruzione stessa, è trasferito nella locazione ISTRUZ.

17. LE DIRETTIVE GENERALI DEL MACRO-11 ASSEMBLER

Le direttive sono frasi che fanno eseguire all'Assembler certe operazioni. A differenza delle istruzioni, le direttive non vengono tradotte in un codice operativo, ma viene eseguita direttamente l'operazione richiesta.
Le direttive Assembler possono essere precedute da una etichetta, salvo casi particolari, e seguite da commenti. Ogni riga può contenere una sola direttiva, che occupa il campo operatore. Il numero e il tipo degli operandi possibili dipende dalla direttiva stessa. Il nome di ogni direttiva inizia con il carattere ".".

17.1. Le direttive .LIST e .NLIST

Può, talora, essere necessario, per economia di tempo (e di carta) sopprimere la lista di un programma o di parti del programma. Per questo scopo esistono due direttive MACRO-11,

.LIST e .NLIST che hanno il seguente formato:

.LIST arg
.NLIST arg

dove "arg" rappresenta una o più opzioni.
Se usate senza argomenti, queste direttive alterano il livello di lista; tale livello è inizializzato a zero e incrementato di uno per ogni direttiva .LIST e decrementato di uno per ogni direttiva .NLIST; quando il livello è negativo viene soppressa la lista del programma sorgente. Quando sono specificate delle opzioni le direttive .LIST e .NLIST non alterano il livello di lista. Gli argomenti possibili sono i seguenti:

argomento	valore default[1]	funzione
SEQ	lista	controlla la lista del numero della riga sorgente. La segnalazione di errore viene generalmente stampata sulla riga precedente lo statement sbagliato;
LOC	lista	controlla la stampa del valore del location counter; questo campo non dovrebbe mai essere soppresso;
BIN	lista	controlla la lista del codice binario generato;
BEX	lista	controlla la lista delle voci binarie successive alla prima voce binaria per ogni statement sorgente;

(1) Per valore "default" si intende il valore che viene assunto automaticamente in mancanza di una specifica particolare.

SRC	lista	controlla la lista del codice sorgente;
COM	lista	controlla la lista dei commenti e può essere usato per ridurre il tempo di stampa e/o lo spazio quando la lista dei commenti non è necessaria;
MD	lista	controlla la lista della definizione delle macro;
MC	lista	controlla la lista delle chiamate delle macro;
ME	non lista	coltrolla la stampa delle espansioni delle macro;
MEB	non lista	controlla la lista del codice binario dell'espansione macro;
CND	lista	controlla la lista delle condizioni non soddisfatte e di tutti gli statements .IF e .ENDC (cfr.§ 18.);
LD	lista	controlla la lista delle direttive .NLIST e .LIST senza argomento;
TOC	lista	controlla la lista della tavola dei contenuti(cfr. §17.2);
TTM	Teletype mode	controlla il formato della lista; se non specificato

		diversamente, la lista viene sempre troncata a 72 caratteri, cioè è fatta in "teletype mode";
SYM	lista	controlla la lista della tavola dei simboli.

Con una stessa direttiva si possono specificare più argomenti; ad esempio la direttiva

.NLIST MC,MD

è equivalente alle due direttive

.NLIST MC
.NLIST MD

Le opzioni di lista possono essere specificate anche con le opzioni o "switches" nella stringa comando per l'Assembler. Questi switches sono:

/LI : arg
/NL : arg

dove "arg" è uno o più degli argomenti specificati per le direttive .LIST e .NLIST.
L'uso degli switches annulla le direttive equivalenti nel programma sorgente.
Per esempio la stringa comando:

#ELIS,LP:/NL:COM:BIN < FOR ØØ1.DAT

sopprime la lista del codice binario e dei commenti; e vengono inoltre ignorate tutte le direttive .LIST con argomenti COM e BIN.
La stringa:

#ELIS,LP:/LI< FORØØ1 .DAT

fa si che vengano ignorate tutte le direttive .LIST e .NLIST senza argomenti, per cui vengono stampate anche le parti di programma sorgente che altrimenti verrebbero soppresse.
Il seguente è un esempio di lista, non in "teletype mode", di un programma in cui si usano le direttive .NLIST SEQ,.NLIST BIN, .NLIST LOC, .LIST ME e .NLIST ME.

```
                 .TITLE CONVERSIONE
             ;
             ; CONVERSIONE DA DECIMALE ASCII A BINARIO
             ; E VICEVERSA
             ;
   000000G       .CSECT CONVE
                 .MCALL .READ,.WRITE,.INIT,.WAIT,.RLSE,.EXIT
                 .MCALL .PARAM,.D2BIN,.BIN2D
                                       .NLIST TTM
                                       .LIST ME               ; LISTA L' ESPANSIONE
                                                              ; DELLA MACRO  .PARAM
   000000                              .PARAM
           000000                  R0=%^00
           000001                  R1=%^01
           000002                  R2=%^02
           000003                  R3=%^03
           000004                  R4=%^04
           000005                  R5=%^05
           000006                  R6=%^06
           000007                  R7=%^07
           000006                  SP=%^06
           000007                  PC=%^07
           177776                  PSW=^O177776
           177570                  SWR=^O177570
                                       .NLIST ME
                                       .GLOBL PRIMO
   000000                          PRIMO: .INIT #LEGGI
   000006                              .INIT #SCRIVI
                                   ;
                                   ; LEGGE UNA SCHEDA CON QUATTRO DATI DECIMALI ASCII
                                   ; SOMMA IL PRIMO CON IL SECONDO
                                   ; E IL TERZO CON IL QUARTO.
                                   ;
   000014                              .READ #LEGGI,#NUM
   000026                              .WAIT #LEGGI
   000034  012701  000440'             MOV #BIN,R1            ; TRASFERISCE IN R1
                                                              ; L' INDIRIZZO DEL
                                                              ; BUFFER BIN CHE
                                                              ; CONTERRA' I DATI
                                                              ; BINARI
                                       .NLIST SEQ             ; SOPPRIME LA STAMPA DEL
                                                              ; NUMERO DI RIGA
   000040  012746  000316'             MOV #DEC1,-(SP)
   000044  012746  000002          CONV: MOV #2,-(SP)
                                                              ; CONVERTE DA DECIMALE
                                                              ; ASCII A BINARIO
```

```
000050  104042                          EMT 42
000052  012621                          MOV (SP)+,(R1)+         ; TRASFERISCE NEL
                                                                ; BUFFER I DATI
                                                                ; BINARI
000054  020127  000450'                 CMP R1,#BIN+10
000060  001371                          BNE CONV
                .NLIST BIN             ; SOPPRIME LA STAMPA DEL
                                       ;CODICE BINARIO
000062          TST  (SP)+
000064          MOV @#BIN,B1
000072          MOV @#BIN+2,B2

000100          ADD B1,B2               ; SOMMA I PRIMI DUE NUMERI
000106          .BIN2D #SUM,B2          ; CONVERTE DA BINARIO A DECIMALE ASCII
                                        ; IN SUM
000124          JSR R5,ZERO
000130          .WORD SUM
000132          ADD BIN+4,BIN+6         ; SOMMA GLI ULTIMI DUE NUMERI
000140          .BIN2D #SUM,BIN+6       ; CONVERTE DA BINARIO A DECIMALE ASCII
                                        ; IN SUM
000156          JSR R5,ZERO
000162          .WORD SUM
000164          .WRITE #SCRIVI,#ASS     ; STAMPA
000176          .WAIT #SCRIVI
000204          .RLSE #LEGGI
000212          .RLSE #SCRIVI
000220          .EXIT
       .NLIST LOC               ; SOPPRIME LA STAMPA DEL LOCATION COUNTER
ZERO:  MOV R1,-(SP)
       MOV R2,-(SP)
       CLR R2
       MOV (R5)+,R1             ; R1 CONTIENE L' INDIRIZZO DEL BUFFER
BLK:   INC R2
       CMPB #060,(R1)
       BNE FINE
       CMP #5,R2
       BEQ FINE
       MOVB #40,(R1)+
       BR BLK
FINE:  MOV (SP)+,R2
       MOV (SP)+,R1
       RTS R5
; DEFINIZIONE DEI FILES DI I/O
       0
```

```
LEGGI : 0
       .RAD50 /INP/
       1
       .RAD50 /BI/
       0
SCRIVI: 0
       .RAD50 /OUT/
       1
       .RAD50 /LP/
NUM: 81.,0,81.
DECI: .BLKB 80.
       .BYTE 12
       .EVEN
BIN:   .BLKW 4
B1:    0
B2:    0
ASS: 130.,0,<STAMP-STAM>
STAM: .BYTE 14,15
       .ASCII /** PROVA DI ADDIZIONE **/
       .BYTE 15,12,12
       .ASCII /   LA SOMMA DEI PRIMI DUE NUMERI E'  /
       .BYTE 15,12
SOM:   .BLKB 5
       .BYTE 15,12,12
       .ASCII /   LA SOMMA DEGLI ULTIMI DUE NUMERI E'  /
       .BYTE 15,12
SUM:   .BLKB 5
       .BYTE 14
STAMP: .BYTE 15,12
       .EVEN
       .END PRIMO
```

Il seguente è invece un esempio di lista in "teletype mode". Come si può vedere le estensioni binarie per le istruzioni che generano più di una voce non sono stampate su una stessa riga, ma su righe successive (una voce per riga).

CONVERSIONE

```
                        .TITLE CONVERSIONE
                ;
                ; CONVERSIONE DA DECIMALE ASCII A BINARIO
                ; E VICEVERSA
                ;
        000000G         .CSECT CONVE
                        .MCALL .READ,.WRITE,.INIT,.WAIT,.RLSE,.EXIT
                        .MCALL .PARAM,.D2BIN,.BIN2D
                        .LIST ME                ; LISTA L' ESPANSIONE
                                                ; DELLA MACRO  .PARAM
   00000                .PARAM
        000000 R0=%^00
        000001 R1=%^01
        000002 R2=%^02
        000003 R3=%^03
        000004 R4=%^04
        000005 R5=%^05
        000006 R6=%^06
        000007 R7=%^07
        000006 SP=%^06
        000007 PC=%^07
        177776 PSW=^0177776
        177570 SWR=^0177570
                        .NLIST ME
                        .GLOBL PRIMO
   00000        PRIMO:  .INIT #LEGGI
   00006                .INIT #SCRIVI
                ;
                ; LEGGE UNA SCHEDA CON QUATTRO DATI DECIMALI ASCII
                ; SOMMA IL PRIMO CON IL SECONDO
                ; E IL TERZO CON IL QUARTO.
                ;
   00014                .READ #LEGGI,#NUM
   00026                .WAIT #LEGGI
   00034 012701         MOV #BIN,R1             ; TRASFERISCE IN R1
         000440'
                                                ; L' INDIRIZZO DEL
                                                ; BUFFER BIN CHE
                                                ; CONTERRA' I DATI
                                                ; BINARI
   00040 012746         MOV #DECI,-(SP)
         000316'
   00044 012746 CONV:   MOV #2,-(SP)
         000002
```

```
                                                  ; CONVERTE DA DECIMALE
                                                  ; ASCII A BINARIO
00050 104042         EMT 42
00052 012621         MOV (SP)+,(R1)+              ; TRASFERISCE NEL
                                                  ; BUFFER I DATI
                                                  ; BINARI
00054 020127         CMP R1,#BIN+10
      000450'
00060 001371         BNE CONV
00062 005726         TST  (SP)+
00064 013767         MOV @#BIN,B1
      000440'
      000356
00072 013767         MOV @#BIN+2,B2
      000442'
      000352
00100 066767         ADD B1,B2                    ; SOMMA I PRIMI DUE NUMERI
      000344
      000344
00106                .BIN2D #SUM,B2-              ; CONVERTE DA BINARIO A DECIMALE ASCII
                                                  ; IN SUM
00124 004567         JSR R5,ZERO
      000072
00130 000566'        .WORD SOM
00132 066767         ADD BIN+4,BIN+6              ; SOMMA GLI ULTIMI DUE NUMERI
      000306
      000306
00140                .BIN2D #SUM,BIN+6            ; CONVERTE DA BINARIO A DECIMALE ASCII
                                                  ; IN SUM
00156 004567         JSR R5,ZERO
      000040
00162 000650'        .WORD SUM
00164                .WRITE #SCRIVI,#ASS          ; STAMPA
00176                .WAIT #SCRIVI
00204                .RLSE #LEGGI
00212                .RLSE #SCRIVI
00220                .EXIT
00222 010146 ZERO:   MOV R1,-(SP)
00224 010246         MOV R2,-(SP)
00226 005002         CLR R2
00230 012501         MOV (R5)+,R1                 ; R1 CONTIENE L' INDIRIZZO DEL BUFFER
00232 005202 BLK:    INC R2
```

```
61 00234 122711        CMPB #060,(R1)
         000060
62 00240 001006        BNE FINE
63 00242 022702        CMP #5,R2
         000005
64 00246 001403        BEQ FINE
65 00250 112721        MOVB #40,(R1)+
         000040
66 00254 000766        BR BLK
67 00256 012602 FINE:  MOV (SP)+,R2
68 00260 012601        MOV (SP)+,R1
69 00262 000205        RTS R5
70               ; DEFINIZIONE DEI FILES DI I/O
71 00264 000000        0
72 00266 000000 LEGGI : 0
73 00270 035200        .RAD50 /INP/
74 00272 000001        1
75 00274 006750        .RAD50 /B1/
76 00276 000000        0
77 00300 000000 SCRIVI: 0
78 00302 060434        .RAD50 /OUT/
79 00304 000001        1
80 00306 046600        .RAD50 /LP/
81 00310 000121 NUM: 81.,0,81.
   00312 000000
   00314 000121
82 00316        DECI:  .BLKB 80.
83 00436    012        .BYTE 12
84                     .EVEN
85 00440        BIN:   .BLKW 4
86 00450 000000 B1:    0
87 00452 000000 B2:    0
88 00454 000202 ASS: 130.,0,<STAMP-STAM>
   00456 000000
   00460 000174
89 00462    014 STAM:  .BYTE 14,15
   00463    015
90 00464    052        .ASCII /** PROVA DI ADDIZIONE **/
   00465    052
   00466    040
   00467    120
   00470    122
   00471    117
   00472    126
   00473    101
```

```
   00474    040
   00475    104
   00476    111
   00477    040
   00500    101
   00501    104
   00502    104
   00503    111
   00504    132
   00505    111
   00506    117
   00507    116
   00510    105
   00511    040
   00512    052
   00513    052
91 00514    015         .BYTE 15,12,12
   00515    012
   00516    012
92 00517    040         .ASCII /   LA SOMMA DEI PRIMI DUE NUMERI E'
   00520    040
   00521    040
   00522    114
   00523    101
   00524    040
   00525    123
   00526    117
   00527    115
   00530    115
   00531    101
   00532    040
   00533    104
   00534    105
   00535    111
   00536    040
   00537    120
   00540    122
   00541    111
   00542    115
   00543    111
```

17.2. Le direttive .TITLE,.SBTTL,.IDENT

All'inizio di ogni pagina della lista viene stampato:

a. il titolo, ricavato dalla direttiva .TITLE

b. la versione dell'Assembler

c. la data

d. l'ora

e. il numero della pagina.

La riga successiva contiene l'eventuale sottotitolo ricavato dalla direttiva .SBTTL.

La direttiva .TITLE serve per dare un nome al modulo oggetto. Tale nome è il primo simbolo dopo la direttiva .TITLE e deve essere formato da caratteri "Radix-50" (ossia le lettere A-Z, le cifre Ø...9 e i caratteri $ e.), fino ad un massimo di sei (i caratteri oltre il sesto sono ignorati).
Per esempio con :

.TITLE PROVA MACRO

viene assegnato il nome PROVA al modulo oggetto del programma assemblato (questo nome è distinto dal nome del file oggetto che compare nella stringa comando).
Se non si usa questa direttiva, al modulo oggetto viene assegnato il nome:

.MAIN.

Se invece in un programma compaiono più direttive .TITLE, il nome assegnato al modulo oggetto è quello specificato dall'ultima. Il nome del modulo compare nella mappa del programma che esegue i collegamenti fra i moduli oggetto (Link).
In programmi lunghi può essere conveniente dare un nome ad ogni sezione di programma. Ciò si ottiene con la direttiva:

.SBTTL

Il testo che compare in questa direttiva serve come identificazione della sezione ed è stampato come seconda riga di ogni pagina della lista. Non c'è limite di caratteri, perché non è il nome di un modulo.
Viene poi stampata automaticamente, a meno che non si usi la direttiva .NLIST TOC (o lo switch /NL:TOC), una tavola di contenuti, che serve da indice in quanto ogni riga contiene il sottotitolo, il numero della pagina e il numero della riga.
Un altro modo per dare un nome al modulo oggetto consiste nell'uso della direttiva .IDENT; con questa direttiva si può specificare una stringa di al più sei caratteri, racchiusi da una coppia di delimitatori, che è passata come parte dell'identificazione del modulo oggetto insieme al nome dato da .TITLE
Se in un programma si trovano più direttive .IDENT, solo l'ultima determina l'identificazione del modulo oggetto.
Per esempio con le direttive:

```
.TITLE PROVA
.IDENT /VØØ1/
```

si dà al modulo oggetto il nome PROVA e l'identificazione VØØ1.

17.3. Il salto di pagina : .PAGE

Dopo 58 righe, il MACRO-11 esegue automaticamente un salto di pa gina. Per avere un salto di pagina prima dei 58 righe, si usa la direttiva:

.PAGE

Questa direttiva non richiede nessun argomento e quando è usata all'interno di un programma causa un salto all'inizio di una nuo va pagina. Se la direttiva .PAGE si trova all'interno di una definizione di macro è ignorata, ma ad ogni chiamata della macro viene effettuato il salto di pagina.

17.4. Le direttive .ENABL e .DSABL

Con le direttive .ENABL e .DSABL è possibile far eseguire certe particolari funzioni al MACRO-11; la funzione desiderata è indicata dall'argomento simbolico, sempre di tre caratteri, che compare nella direttiva. La forma di queste due direttive è:

.ENABL arg
.DSABL arg

dove"arg"è uno dei seguenti argomenti possibili:

ABS l'attivazione di questa funzione produce un output binario assoluto; in mancanza di una attivazione esplicita viene assunto il caso .DSABL ABS;

AMA quando attivata, questa funzione fa si che tutti gli indirizzi relativi siano assemblati come indirizzi assoluti;

CDR se in un programma compare la frase .ENABL CDR le colonne dalla 73 in poi sono considerate come commenti, anche se non iniziano con ";";

FPT se questa funzione è attivata i numeri in floating point vengono troncati, piuttosto che arrotondati, come avviene normalmente;

LC se questa funzione è attivata l'Assembler accetta le lettere minuscole in ingresso invece di convertirle in lettere maiuscole;

LSB con .ENABL LSB si inizia un blocco di simboli locali, che non termina finché non si incontra una direttiva .CSECT o una etichetta simbolica dopo la direttiva .DSABL LSB. Il caso default è .DSABL LSB ;

PNC lo statement .DSABL PNC sopprime la stampa del codice binario;il caso default è: .ENABL PNC ;

REG questa funzione definisce i nomi standard per i registri, ossia

```
RO = %Ø
 :
PC = %7
```

GBL se questa funzione è attivata l'Assembler tenta di risolvere i riferimenti globali indefiniti.

Queste funzioni possono essere controllate anche con gli switches nella stringa comando per il MACRO Assembler. Questi switches sono:

```
/EN:arg
/DS:arg
```

dove "arg" è uno dei parametri possibili. L'uso degli switches sopprime l'attivazione o disattivazione di tutte le occorrenze dell'argomento nel programma.

17.5. La direttiva .BYTE

Per generare bytes successivi di dati si usa la direttiva .BYTE, che ha la forma:

```
                .BYTE exp
oppure          .BYTE exp1,exp2,...
```

Il valore dell'espressione deve essere assoluto e rappresentabile in 8 bits. Il valore a 16 bits dell'espressione deve essere, cioè, tale che il byte di sinistra contenga tutti zero o tutti 1. Ciascuna espressione operando è memorizzata in un byte del programma oggetto; gli operandi multipli sono separati da virgola e memorizzati in bytes successivi.
Per esempio:

```
.=666
.BYTE 15,12
```

il valore 15_8 è memorizzato nella locazione 666 e nella 667 viene memorizzato il valore 12_8.
Se il valore della espressione non è rappresentabile in 8 bits, viene troncato e viene segnalato un errore con codice T.
Se un operando è assente viene interpretato come zero. Pertanto:

```
.=721
.BYTE  , ,
```

genera tre bytes uguali a zero alle locazioni 721, 722, 723.

17.6. La direttiva .WORD

La direttiva .WORD è usata per generare parole successive di dati. Il formato è:

```
.WORD exp
:WORD expl,exp2,...
```

Il valore delle espressioni deve essere rappresentabile in 16 bits, altrimenti è troncato e viene segnalato un errore con codice T. Per esempio:

```
.WORD Ø, 177777, AB
```

genera tre parole successive in cui viene rispettivamente memorizzato Ø, 177777 e il valore di AB. Se un operando è assente, viene interpretato come zero.
Se il campo operatore di uno statement è vuoto, viene interpretato implicitamente come una direttiva .WORD .
L'uso di questa convenzione è però sconsigliato in quanto si prevede che non sarà consentito nelle future versioni dell'Assembler PDP-11.
Il primo termine della prima espressione non deve essere un nome simbolico di una istruzione o direttiva assembler a meno che non sia preceduto da un operatore + o -.
Per esempio:

```
          .=33Ø
LABEL:    +MOV, LABEL
```

memorizza nella locazione 33Ø Ø1ØØØØ (codice operativo di MOV) e il valore 33Ø nella locazione 332.

17.7. Le direttive .FLT2 e .FLT4

Analogamente alla direttiva .WORD, le direttive .FLT2 e .FLT4 servono per memorizzare dati in floating point durante l'assemblaggio. Queste direttive hanno la forma:

```
.FLT2 arg1, arg2,...
.FLT4 arg1, arg2,...
```

dove arg1,arg2,... rappresentano uno o più dati in floating point, separati da virgola.
Un numero in floating point è rappresentato da una stringa di cifre decimali; questa stringa può contenere un punto decimale e può essere seguita da un indicatore di esponente nella forma della lettera E e un esponente decimale con segno.
Sono quindi tutte possibili le seguenti rappresentazioni dello stesso numero in floating point:

```
1
1.Ø
1.
1.ØEØ
1EØ
.1E1
1ØE-1    ecc.
```

Se il numero è preceduto da un segno meno, viene fatto il complemento del bit del segno (si ricorda infatti che i numeri in floating point negativi non sono rappresentati nella forma di complemento a due).
Normalmente, a meno cioè che non si usi .ENABL FPT, i numeri in floating point sono arrotondati e non troncati; ossia quando un numero eccede i limiti del campo in cui deve essere memorizzato, il bit di ordine più alto in eccesso è aggiunto all'ultimo bit del campo.
Con la direttiva .FLT2 ciascun argomento viene memorizzato in due parole, mentre con .FLT4 viene memorizzato in quattro parole.

Esempio

I numeri Ø.1 e Ø.5 possono essere così assemblati:

```
Ø37314  146315      FLOA: .FLT2 1.E-1
Ø4ØØØØ  ØØØØØØ            .FLT2 5.E-1
```

I numeri a sinistra sono il contenuto (ottale) delle voci generate.

17.8. La conversione: .ASCII,.ASCI2

Per memorizzare in una parola uno o due caratteri alfanumerici si usano i caratteri ' (apice) e " (virgolette).
L'apice seguito da *un solo* carattere genera una parola che contiene nel byte di destra la rappresentazione ottale del carattere ASCII e nel byte di sinistra zero.
Per esempio:

```
MOV  #'A,BUFF
```

genera la seguente parola a 16 bits, che sarà trasferita in BUFF,:

ØØØØØØØØ	Ø1ØØØØØ1

(il valore ottale ASCII del carattere A è $1Ø1_8$)
Il carattere " seguito da *due* caratteri genera una parola in cui è memorizzata la rappresentazione ASCII in 7 bits.
Il byte di destra contiene la rappresentazione del primo carattere, il byte di sinistra contiene la rappresentazione del secondo.
Per esempio:

```
MOV  #"AB,BUFF
```

genera la seguente parola, da trasferire in BUFF:

Ø1ØØØØ1Ø	Ø1ØØØØØ1

(il valore ottale ASCII di B è 1Ø2$_8$).

Sia l'apice che le virgolette non possono essere seguiti da ritorni di carrello, caratteri nulli, né controlli di fine riga o fine pagina.

Per convertire una stringa di caratteri nel loro equivalente ASCII a 7 bits si usano le direttive

.ASCII e .ASCIZ

Il formato della direttiva .ASCII è:

.ASCII /stringa di caratteri/

dove / / sono due caratteri delimitatori e possono essere due caratteri qualsiasi esclusi < ; e = e ogni carattere all'interno della stringa.

I caratteri ; e = non sono, in realtà, incorretti, ma il loro uso è sconsigliato perché possono indurre in errori di interpretazione. Infatti per esempio in:

.ASCII /ABC/;DEF;

;DEF; è trattato come un commento e ignorato; mentre in:

.ASCII = DEF =

viene fatto l'assegnamento:

.ASCII = DEF

ed è segnalato un errore di tipo Q per la presenza del secondo =.
La stringa di caratteri non può contenere né caratteri nulli, né ritorni di carrello né controlli di fine riga o fine pagina.
Questi caratteri particolari possono essere espressi in cifre, della base numerica corrente, e racchiusi fra parentesi acute.
Per esempio:

```
.ASCII  /FI/ <15><12>/ITALY/
```

memorizza in bytes consecutivi: F,I,15 (ritorno di carrello), 12 (fine riga), I, T, A, L,Y.

```
.ASCII  /<ABC>/
```

memorizza in bytes successivi: < ,A,B,C,> .
Ricordando, poi che 1Ø1$_8$ è la rappresentazione ottale ASCII di A, si ha che:

```
.ASCII  <1Ø1>
```

è equivalente a .ASCII /A/.

La direttiva .ASCIZ è equivalente a .ASCII soltanto inserisce automaticamente un byte uguale a zero alla fine della stringa.
Pertanto se una stringa è generata con .ASCIZ, la ricerca del byte nullo determina la fine della stringa.
E' bene tener presente che il carattere "spazio" (blank) differisce da quello nullo in quanto il primo ha rappresentazione ottale ASCII Ø4Ø, mentre il secondo ØØØ.

```
PROVA ASCII

     1                                          .TITLE PROVA ASCII
     2                                          .MCALL .EXIT
     3          000015                          CR=15
     4          000012                          LF=12
     5 000000                           INIZIO:
     6 000000   012701  000054'                 MOV #CAR,R1
     7 000004   012702  000016'                 MOV #CAR1,R2
     8 000010   112122                  VB:     MOVB (R1)+,(R2)+
     9 000012   001376                          BNE VB
    10 000014                                   .EXIT
    11 000016                           CAR1:   .BLKB 29.
    12                                          .EVEN
    13 000054     015      012     124  CAR:    .ASCIZ <CR><LF>/TRASFERIMENTO CARATTERI/<CR><LF>
       000057     122      101     123
       000062     106      105     122
       000065     111      115     105
       000070     116      124     117
       000073     040      103     101
       000076     122      101     124
       000101     124      105     122
       000104     111      015     012
       000107     000
    14                                          .EVEN
    15          000000'                         .END INIZIO
```

Fig. 13 - Uso della direttiva .ASCIZ

17.9. La direttiva .RAD5Ø

La direttiva .RAD5Ø permette di manipolare dati codificati nella forma Radix-5Ø. Questa forma permette di rappresentare 3 caratteri per voce; i caratteri rappresentabili sono: le lettere da A e Z, le cifre da Ø a 9, il simbolo $ e il punto (.).
Il formato della direttiva è:

.RAD5Ø /stringa/

dove /e/ sono due caratteri delimitatori esclusi =, < e ;
e "stringa" è una lista di caratteri da convertire.
Se i caratteri sono meno di tre (o l'ultimo insieme contiene meno di tre caratteri), sono rappresentati più a sinistra possibile nella parola (left justified), seguiti da spazi bianchi.
Per esempio:

.RAD5Ø /ABCD/

genera due parole: nella prima memorizza l'equivalente di ABC e nella seconda D ɓ ɓ.
Ciascun carattere è tradotto nel suo equivalente Radix-5Ø, come indicato nella tabella seguente:

carattere	equivalente ottale Radix-5Ø
spazio (ɓ)	Ø
A - Z	1 - 32
$	33
.	34
Ø - 9	36 - 47

Il valore Radix-5Ø di tre caratteri C1,C2,C3 è calcolato nel modo seguente:

((C1*5Ø)+C2)*5Ø+C3

Per esempio il valore Radix-5Ø di ABC è:

((1*5Ø)+2)*5Ø+3 = 3223

(cfr. Appendice A)

17.10. Il controllo della base di numerazione: .RADIX,↑D,↑O,↑C,↑F,↑B.

I numeri in un programma sorgente in MACRO-11 sono considerati ottali. Tuttavia l'utente ha la possibilità di dichiarare numeri in una delle seguenti basi:

2, 4, 8, 1Ø

Ciò si ottiene con la direttiva .RADIX, che ha il seguente formato:

.RADIX n

dove n è una delle basi consentite. Il numero n è sempre interpretato come decimale. I numeri che compaiono nelle istruzioni che seguono la direttiva .RADIX n sono interpretati tutti in base n, finché non si trova un'eventuale altra direttiva .RADIX m, con $n \neq m$.

Pertanto se all'inizio di un programma si trova la direttiva .RADIX 1Ø, e non vi sono altre direttive .RADIX, tutti i numeri del programma sorgente sono interpretati come decimali.

Poiché la base assunta in mancanza di una specifica è la base ottale, per passare da una sezione in una base qualsiasi ad una sezione in base ottale è sufficiente usare

.RADIX

invece di

.RADIX 8

Talora, però, può essere necessario usare solo per un termine o un numero una base diversa da quella della sezione di codice in cui si trova. Per far questo esistono tre operatori unari: ↑D, ↑B, ↑O per cui:

↑Dx   x è trattato come un numero decimale

↑Bx   x è trattato come un numero binario

↑Ox   x è trattato come un numero ottale.

Per esempio:

```
MOV  #24,RØ
MOV  #↑D24,R1
MOV  #24,R2
```

con la prima istruzione viene trasferito in RØ il valore 24 ottale mentre con la seconda viene trasferito in R1 il valore 24 decimale senza però che venga cambiata la base di numerazione nelle istruzioni seguenti, per cui in R2 viene trasferito 24 ottale.
La freccia e la lettera che indica la base non possono essere separate da uno spazio, mentre il numero può essere separato dall'operatore unario; quando un termine o una espressione deve essere interpretato in un altra base, va racchiuso tra parentesi acute. Per esempio:

↑D123 definisce il numero decimale 123

↑B ØØØ11 definisce il numero binario 11

↑O <A+3> definisce come ottale il termine A + 3

Per dichiarare un numero decimale si può usare, oltre all'operatore unario ↑D, il punto decimale. Sono cioè considerati decimali, per esempio,

1ØØ. (144$_8$)

828. (1474$_8$)

128. (2ØØ$_8$)

Per specificare un numero in floating point, memorizzato in una sola voce, si usa l'operatore unario ↑F.
Per esempio:

FLOA: ↑F 2.7

crea una parola alla locazione FLOA che contiene il valore 2.7 nel seguente formato:

	7	6 Ø
S	caratteristica	mantissa

Questa parola è la prima delle due parole che contengono un numero in floating point.
L'argomento dell'operatore ↑F non può essere una espressione e deve avere lo stesso formato degli argomenti delle direttive .FLT2 e .FLT4.
Infine l'operatore ↑C serve per ottenere il complemento a 1 dell'operando. Ad esempio:

COMPL: ↑C123456

memorizza nella locazione COMPL il complemento a 1 di 123456, ossia Ø54321.
Poiché tutti questi operatori sono unari, i loro argomenti possono essere termini e gli operatori stessi possono essere ripetuti ricorsivamente. Ad esempio:

↑C↑D25

è equivalente a ↑C31.
Il termine creato dall'operatore unario e dal suo argomento può esso stesso essere usato in una espressione. Per esempio:

↑C2 + 3 è equivalente a : ⟨↑C2⟩ + 3.

17.11. Il controllo del location counter:.EVEN,.ODD,.BLKB,.BLKW

Le quattro direttive che controllano il movimento del location counter sono: .EVEN, .ODD, .BLKB, .BLKW.
La direttiva .EVEN assicura che il location counter contenga un *indirizzo di memoria pari*, aggiungendo uno se l'indirizzo corrente è dispari. Se il valore corrente è pari non viene eseguita nessuna azione. La direttiva .EVEN non ha argomenti e può essere usata come segue:

```
.ASCII  /TESTO/
.EVEN
```

in questo modo si assicura che l'istruzione successiva si trovi ad un indirizzo pari, cioè ad un inizio di parola.
La direttiva .ODD, invece, assicura che il location counter sia *dispari*, aggiungendo uno se il valore corrente è pari.

Le due direttive .BLKB e .BLKW servono per riservare blocchi di memoria. La .BLKB serve per riservare blocchi di memoria misurati *in bytes*, mentre la .BLKW riserva blocchi di memoria misurati

in parole. Il loro formato è:

```
.BLKB  exp.
.BLKW  exp
```

dove exp è il numero di bytes o parole che deve essere riservato. Se non è presente nessun argomento viene assunto il valore 1. L'espressione deve essere completamente definita al momento dell'assemblaggio.
Per esempio:

```
BUFF: .BLKW 3
```

riserva 3 parole, di cui la prima si trova all'indirizzo BUFF ;

```
BUFF1: .BLKB 7
```

riserva 7 bytes di cui il primo si trova all'indirizzo BUFF1.
La direttiva .BLKB exp ha lo stesso effetto di:

```
. = .+exp
```

ma è chiaramente più facile da interpretare nel contesto del codice sorgente.

17.12. .END

La direttiva .END indica la fine fisica del programma sorgente. Il formato di questa direttiva è:

```
.END exp
```

dove exp è un parametro, facoltativo, che indica il punto di ingresso del programma.

Quando il modulo eseguibile è caricato l'esecuzione inizia al punto d'ingresso specificato in questa direttiva.

17.13. .LIMIT

La direttiva .LIMIT riserva due parole nelle quali il LINK mette, rispettivamente, l'indirizzo più basso e più alto di memoria occupata dal modulo rilocabile. L'indirizzo più basso, messo nella prima parola, è l'indirizzo del primo byte di codice, mentre l'indirizzo più alto è l'indirizzo del primo byte che segue il codice rilocabile, cioè del primo byte non occupato dal modulo.

17.14. .PSECT, .CSECT, .ASECT

E' possibile dividere un programma in sezioni, fino ad un massimo di 255 sezioni: una sezione assoluta, una sezione rilocabile senza nome e 253 sezioni rilocabili con un nome.
Per creare queste sezioni si usa la direttiva .PSECT, che inoltre permette di spartire i dati fra più moduli.
Il formato di questa direttiva è:

```
.PSECT  [nome] [,RO/RW] [,I/D] [,GBL/LCL] [,ABS/REL]
        [,CON/OVR] [,HGH/LOW]
```

dove [ ] indica che il nome o i parametri sono opzionali.
Il significato degli attribuiti è specificato nella tabella seguente dove è indicato anche il valore default, ossia il valore assunto in mancanza di una specifica diversa.

parametro	valore default	descrizione
RO/RW	RW	definisce il tipo di accesso permesso alla sezione di programma; RO significa solo lettura, mentre RW significa lettura e scrittura;
I/D	I	differenzia i simboli globali che sono punti di ingresso, (I), dai simboli globali che sono dati (D);
GBL/LCL	LCL	definisce il rango in cui la P-sezione è considerata dal LINK: GBL significa globale e la P-sezione sarà considerata oltre i limiti del segmento (overlay) in cui è definita; LCL significa locale, e la P-sezione è considerata solo all'interno del segmento in cui è definita; se il programma è formato da un solo segmento, questo attributo è ignorato;
ABS/REL	REL	specifica se la sezione è assoluta o rilocabile; quando la sezione è rilocabile il LINK aggiunge l'indice di rilocazione a tutti gli indirizzi nella sezione;
CON/OVR	CON	specifica l'allocazione della sezione; CON significa che tutti i riferimenti alla sezione da moduli diversi sono concatenati per formare l'allocazione totale della sezione; OVR indica che tutte

parametro	valore default	descrizione
		le allocazioni riferite da altri moduli sono sovrapposte; così la allocazione totale della sezione è determinata dalla più grande richiesta fatta dai singoli moduli;
HGH/LOW	LOW	specifica il tipo di memoria da usare per la sezione di programma; HGH: memoria ad alta velocità. LOW: memoria a bassa velocità; attualmente questo attributo è ignorato.

Ad un programma senza direttiva .PSECT, viene assegnato il nome .MAIN., e tutti gli attributi default.
Il primo parametro deve essere un nome, cioè da 1 a 6 caratteri in formato RADIX-50; se il nome viene omesso, al suo posto deve essere usata una virgola. Per esempio:

```
.PSECT  ,ABS
```

è una direttiva .PSECT senza nome e con il solo parametro che specifica una sezione assoluta. Per gli altri parametri viene assunto il valore default.
Una volta definiti certi attributi per una .PSECT col nome, il MACRO Assembler ritiene validi gli stessi attributi per tutte le .PSECT seguenti con lo stesso nome. Per esempio una sezione può essere specificata come:

```
.PSECT  GAMMA, RO, I, ABS, OVR
```

la stessa sezione di programma può in seguito essere riferita con

.PSECT GAMMA

e saranno ancora validi gli stessi attributi.
Poiché l'Assembler usa per ogni sezione un contatore di locazione, l'utente può scrivere statements che non sono fisicamente contigui ma che sono caricati consecutivamente, come mostrato dal seguente esempio:

```
                                          .TITLE SEZIONI
000000                             PRIMO:
          000000'                         .PSECT UNO
000000    010046                          MOV R0,-(SP)
000002    010146                          MOV R1,-(SP)
000004    010246                          MOV R2,-(SP)
000006    005000                          CLR R0
000010    005001                          CLR R1
000012    005002                          CLR R2
000014    012702   000010                 MOV #10,R2
000020    012700   000044'                MOV #A,R0
000024    012701   000064'                MOV #B,R1
          000000'                         .PSECT DUE,ABS
          177776                          PS=177776
          000004'                         .=.+4
000004    000000G                         .WORD EXT
          000030'                         .PSECT UNO
000030    012021                   AB:    MOV (R0)+,(R1)+
000032    077202                          SOB R2,AB
000034    012602                          MOV (SP)+,R2
000036    012601                          MOV (SP)+,R1
000040    012600                          MOV (SP)+,R0
                                          .MCALL .EXIT
000042                                    .EXIT
000044                             A:     .BLKW 10
000064                             B:     .BLKW 10
          000000'                         .END PRIMO
```

Fig.14 - Uso della direttiva .PSECT

Alla prima occorrenza di una direttiva .PSECT con un dato nome, al location counter è dato il valore zero, rilocabile o assoluto. Il rango di ciascuna direttiva si estende finché un'altra direttiva inizia una nuova sezione. Ulteriori occorrenze di uno

stesso nome di sezione in successive direttive .PSECT riprendono l'assemblaggio al punto dove era terminata la sezione precedente. Le etichette in una sezione assoluta sono assolute, mentre sono rilocabili in una sezione rilocabile.
Le sezioni con lo stesso nome e con l'attributo OVR sono tutte caricate alla stessa locazione dal LINK, e quindi possono essere usate per ridefinire le stesse sezioni di memoria.
I nomi di sezioni possono essere ridefiniti come simboli interni al programma, ma non possono essere usati come nomi .GLOBL.
Sebbene la direttiva .PSECT assommi tutte le possibilità delle direttive .ASECT e .CSECT definite per altri Assembler PDP-11, per compatibilità con i programmi non scritti in DOS/BATCH MACRO, il MACRO Assembler accetta le direttive .ASECT e .CSECT, ma le assembla come se fossero direttive .PSECT con gli attributi default illustrati nella tabella seguente:

attributo	.ASECT	.CSECT (con nome)	.CSECT
nome	ABS	nome	vuoto
accesso	RW	RW	RW
tipo	I	I	I
rango	GBL	GBL	LCL
rilocazione	ABS	REL	REL
allocazione	OVR	OVR	CON
memoria	LOW	LOW	LOW

Il formato delle direttive .ASECT e .CSECT è:

.ASECT

.CSECT [simbolo]

Per esempio:

.CSECT COM

è equivalente a:

.PSECT COM,RW,I,GBL,REL,OVR,LOW

Da quanto detto appare chiaro che le sezioni di programma con un nome e rilocabili operano come le frasi COMMON etichettato del FORTRAN. Pertanto si possono usare delle direttive .CSECT con nome (o l'equivalente .PSECT) per realizzare il passaggio di parametri fra moduli oggetto e quindi anche fra un programma FORTRAN e una routine in linguaggio assemblativo.

17.15 .GLOBL

Il risultato dell'assemblaggio è un modulo oggetto rilocabile e un file contenente la lista e la tavola dei simboli. Il LINK unisce i moduli assemblati separatamente in un modulo eseguibile; i moduli oggetto (se sono più di uno) sono uniti tramite i simboli globali, così che in un modulo si può fare riferimento ad un simbolo globale definito (come etichetta o con un assegnamento diretto) in un altro modulo.
I simboli globali sono definiti, unicamente, con la direttiva .GLOBL, che ha il formato:

.GLOBL simb1,simb2,...

dove simb1,simb2 sono nomi simbolici, separati da virgola o da uno spazio.
I simboli che appaiono in una direttiva .GLOBL sono definiti all'interno del programma, oppure sono simboli esterni definiti in un altro programma; questo programma è unito dal LINK al programma corrente prima dell'esecuzione al fine di risolvere tut-

ti i riferimenti a simboli globali.
Tutti i simboli globali interni al programma devono essere definiti alla fine del primo passo dell'assemblaggio.
Un simbolo esterno può essere usato come operando in una istruzione o come argomento di una direttiva, ma non può essere usato nella valutazione di una espressione in una frase di assegnamento diretto. Un simbolo globale definito all'interno del programma può essere usato anche nella valutazione di una espressione.

18. I BLOCCHI CONDIZIONALI

La forma di un blocco condizionale, ossia di un blocco di codice sorgente che in base al valore di una certa condizione è incluso o ignorato nel processo di assemblaggio, è la seguente:

```
.IF  cond, argomento/i
       :
(blocco condizionale)
.ENDC
```

dove .IF è la direttiva che apre il blocco condizionale;
cond è la condizione che deve essere soddisfatta perché il blocco sia incluso nell'assemblaggio
argomento è una funzione della condizione da verificare
.ENDC è la direttiva che chiude il blocco condizionale.
I blocchi condizionali possono essere inseriti uno dentro l'altro fino ad un livello massimo di 16.

Le condizioni e i relativi argomenti sono illustrati nella tabella seguente:

condizioni (positive)	condizioni (complementari)	argomenti	il blocco è assemblato se
EQ	NE	espressione	espressione=Ø (o ≠ Ø)
GT	LE	" "	espressione >Ø (o ≤ Ø)
LT	GE	" "	espressione <Ø (o ≥ Ø)
DF	NDF	argomento simbolico	simbolo è definito (o indefinito)
B	NB	argomento tipo macro	argomento è vuoto (o non vuoto)
IDN	DIF	due argomenti tipo macro separati da virgola	argomenti identici (o differenti)
Z	NZ	espressione	lo stesso di EQ/NE
G	L	" "	lo stesso di GT/LE

Un argomento di tipo macro è un argomento racchiuso in parentesi acute, o dalla costruzione particolare con il carattere ↑ .
Per esempio:

⟨A,B,C⟩ o ↑/123/

Esempio

```
.IF  EQ  ALFA
  .
  .
  .
.ENDC
```

il blocco è assemblato se ALFA = Ø.
Con le condizioni DF e NDF si possono usare due operatori binari che permettono di raggruppare gli argomenti, e precisamente:

& operatore di AND logico

! operatore di OR inclusivo.

Per esempio:

```
.IF  DF  ARG1 & ARG2
 .
 .
 .
.ENDC
```

il blocco è assemblato se sono definiti entrambi gli argomenti ARG1 e ARG2.

18.1. <u>Sottoblocchi condizionali</u>

I sottoblocchi condizionali possono essere messi all'interno di blocchi condizionali per modificare l'effetto del test sulla condizione d'entrata nel blocco.
Le direttive che definiscono i sottoblocchi sono:

.IFF il codice che segue questo statement fino al successivo sottoblocco condizionale o fino allo fine del blocco condizionale è incluso nel programma se la condizione d'entrata nel blocco è falsa;

.IFT come sopra, con la differenza che la condizione d'entrata sia vera;

.IFTF il codice che segue questo statement è incluso nel programma qualsiasi sia il valore della condizione d'entrata.

L'argomento implicito di una direttiva sottocondizionale è il valore della condizione d'entrata nel blocco condizionale.

Per esempio:

```
.IF  DF  ARG1
.IFF
.MOV  R1,R2
.IFT
.MOV  R2,R3
.IFTF
.MOV R3,R4
.ENDC
```

l'istruzione MOV R1,R2 è assemblata se ARG1 è indefinito, mentre l'istruzione MOV R2,R3 è assemblata se ARG1 è definito.
Infine, l'istruzione MOV R3,R4 è comunque assemblata.

18.2. I condizionali immediati

Le direttive che definiscono dei condizionali immediati sono un mezzo per scrivere un blocco condizionale in una sola riga.
Non è quindi più necessaria la direttiva .ENDC.
I condizionali immediati sono della forma:

```
.IIF  cond,arg,statement
```

dove:

cond — è una delle condizioni definite per i blocchi condizionali

arg — è l'argomento associato con la condizione specificata

statement — è la frase da eseguire se la condizione è verificata.

Per esempio:

```
        .IIF  EQ,B,BEQ BETA
```

genera il codice BEQ BETA se B = ∅.
Uno statement .IIF *non* deve contenere una etichetta; ogni eventuale etichetta può essere messa sulla riga precedente.
Per esempio:

```
        LABEL:
              .IIF EQ,B,BEQ ALFA
```

o inclusa come parte dello statement condizionale:

```
              .IIF EQ,B,LABEL:BEQ,ALFA
```

18.3. Le direttive condizionali del PAL-11R

Per compatibilità con i programmi scritti sotto PAL-11R, sono consentite le seguenti direttive condizionali, che si usano nello stesso modo delle direttive condizionali MACRO.

direttiva	argomenti	il blocco è assemblato se
.IFZ o .IFEQ	espressione	espressione = ∅
.IFNZ o .IFNE	" "	espressione ≠ ∅
.IFL o .IFLT	" "	" " < ∅
.IFG o .IFGT	" "	" " > ∅
.IFLE	" "	" " ≤ ∅
.IFGE	" "	" " ≥ ∅
.IFDF	espressione logica	espressione è vera (definita)
.IFNDF	" "	" " " falsa(non definita)

19. LE DIRETTIVE MACRO

19.1. La definizione delle macro

Nella stesura di programmi può capitare che si debba ripetere più volte una stessa sequenza di istruzioni. E' quindi molto utile poter definire questa sequenza una volta per tutte e fare un semplice riferimento ad essa tutte le volte che sia necessario, eventualmente specificando ogni volta i parametri su cui deve operare.

Queste sequenze di istruzioni sono dette "macro". Una volta che una macro è stata definita, è richiamata con un solo statement che contiene il nome e gli eventuali parametri attuali. Il programma è quindi tradotto effettuando delle espansioni locali ossia sostituendo l'intera sequenza di istruzioni al posto del riferimento alla macro.

La prima frase di una definizione di macro è una direttiva .MACRO con il seguente formato:

```
.MACRO  nome, lista di parametri formali
```

dove:

nome	è il nome che identifica la macro. Il nome è un simbolo MACRO-11 e può essere usato anche come etichetta all' interno del programma
lista di parametri formali	rappresenta uno o più simboli che possono apparire dovunque nel corpo della macro, perfino come etichette. Questi simboli possono essere usati dovunque nel programma; se i parametri formali sono più di uno, sono separati da un carattere separatore, generalmente da virgola.

Tra il nome della macro e la lista di parametri c'è un carattere separatore, generalmente uno spazio o una virgola. In una frase .MACRO non deve esserci etichetta, mentre i commenti possono se-

guire la lista di parametri formali.
Per esempio:

```
.MACRO  MAX,A,B  ;  PONE IN A IL MASSIMO TRA A E B
```

La frase che indica la fine della definizione della macro è la direttiva .ENDM nelle due forme:

```
.ENDM
```

oppure

```
.ENDM  nome
```

dove "nome" è il nome della macro chiusa da questa frase. Se il nome è presente nella direttiva .ENDM, deve essere lo stesso che compare nella direttiva .MACRO.
La statement .ENDM può contenere un commento ma non può avere etichette.
Può darsi che una definizione di macro richieda più di un punto di uscita; per implementare questa possibilità è prevista la direttiva .MEXIT .La direttiva .MEXIT termina l'espansione della macro corrente esattamente come se si fosse incontrata una direttiva .ENDM .

19.2. La chiamata delle macro

Una macro è richiamata tramite il nome e una eventuale lista di parametri attuali. Quindi per esempio la macro MAX è chiamata dallo statement:

```
MAX MAX,COM
```

Nella relativa espansione locale ai parametri formali A e B vengono sostituiti i parametri attuali MAX e COM. Lo statement che contiene una chiamata di una macro può essere etichettato. Se i parametri attuali sono più di uno, devono essere separati da un

carattere separatore.
Se un parametro contiene caratteri separati deve essere racchiuso in parentesi acute; si può usare anche la costruzione con ↑, che permette quindi di passare come parametri anche le parentesi acute.
Per esempio:

```
.MACRO  ISTR,A,B,C
:
ISTR  <CLR R1>,#1Ø,C
```

con questa chiamata l'intera istruzione:

```
CLR R1
```

viene sostituita a tutte le occorrenze del parametro A.
Si poteva usare anche la costruzione:

```
ISTR ↑\CLR R1\,#1Ø,C
```

Può darsi che una macro richiami al suo interno un'altra macro.
Per passare un parametro alle macro richiamate da altre macro, il parametro deve essere racchiuso tra parentesi acute tante volte quanti sono i livelli di chiamate.
Per esempio:

```
.MACRO UNO,PAR1,PAR2
DUE PAR1
DUE PAR2
.ENDM UNO

.MACRO DUE PAR3
PAR3
ADD #6,RØ
.ENDM DUE
```

```
UNO <<MOV RØ,R1>>,<<CLR RØ>>
DUE <MOV RØ,R1>
MOV RØ,R1
ADD #6,RØ
DUE <CLR RØ>
CLR RØ
ADD #6,RØ
```

Se una macro è definita all'interno di un'altra macro, la macro più interna non può essere richiamata, finché non sia stata chiamata almeno una volta la macro che la contiene.

L'apice, ('), opera come un carattere separatore in una definizione della macro. Un apice che precede e/o segue un parametro formale in una definizione di macro è rimosso e a quel punto avviene la sostituzione del parametro attuale.
Per esempio:

```
.MACRO SALVA I
 MOV R'I, -(SP)
.ENDM

 SALVA Ø
 MOV RØ,-(SP)

 SALVA 1
 MOV R1,-(SP)
     :
```

I parametri costituiti da più caratteri ma senza spazi bianchi, ";", e virgola possono essere trasmessi senza racchiunderli in parentesi. Per esempio :

```
.MACRO PUSH PAR
MOV PAR, -(SP)
.ENDM

PUSH 2(%3)
```

genera il seguente codice.

```
MOV 2(%3),-(SP)
```

Un parametro preceduto dall'operatore unario \ è trattato come un numero nella base corrente.
Se il numero dei parametri attuali supera quello dei parametri formali, i parametri attuali in eccesso sono ignorati; se invece i parametri attuali sono meno dei parametri formali, i parametri mancanti sono considerati nulli, cioè non consistenti di alcun carattere.

Il MACRO-Assembler può creare simboli locali della forma n$, con $64 < n < 127$. Questi simboli locali creati automaticamente sono particolarmente utili quando nella espansione della macro è richiesta una etichetta. Tale etichetta può essere inserita fra i parametri formali e quindi cambiata ad ogni chiamata della macro; altrimenti viene generata la stessa etichetta ad ogni espansione, e quindi tale etichetta può risultare multidefinita . A meno che l'etichetta non sia riferita dal di fuori della macro, non c'è ragione perché il programmatore tenga conto delle etichette.
Per generare i simboli locali si inserisce l'etichetta fra i parametri formali preceduta, però da ? ; al momento della chiamata non si specifica il parametro attuale corrispondente all'etichetta. I simboli locali, infatti, sono generati solo dove i parametri attuali nella chiamata della macro sono o nulli o mancanti. Se nella chiamata della macro è specificato il parametro formale, il simbolo locale non viene generato ed è eseguita una normale sostituzione.

```
        .CSECT DEF
        .MACRO BINDE BUFF,WORD,?MENO,?CONVE
        .MCALL .BIN2D
        TST WORD
        BMI MENO
        MOVB #040,BUFF
        BR CONVE
MENO:   NEG WORD
        MOVB #055,BUFF
CONVE:  .BIN2D #BUFF+1,WORD
        .ENDM BINDE



        .LIST ME
        BINDE DEC,OTT
        .MCALL .BIN2D
        TST OTT
        BMI 66$
        MOVB #040,DEC
        BR 67$
66$:    NEG OTT
        MOVB #055,DEC
67$:    .BIN2D #DEC+1,OTT


        BINDE DECI,OTT+2
        .MCALL .BIN2D
        TST OTT+2
        BMI 68$
        MOVB #040,DECI
        BR 69$
68$:    NEG OTT+2
        MOVB #055,DECI
69$:    .BIN2D #DECI+1,OTT+2


DEC:    .BLKB 6
DECI:   .BLKB 6



OTT:    .BLKW 4
        .END INIZIO
```

Fig. 15 - Generazione automatica dei simboli locali

19.3. .NARG,.NCHR,.NTYPE

La direttiva .NARG serve per determinare il numero di parametri in una chiamata di macro; il formato è:

.NARG simbolo

La direttiva .NCHR serve per determinare il numero di caratteri in una stringa. Il formato è:

.NCHR simbolo, <stringa>

Il simbolo, separato con un carattere separatore dalla stringa, è uguagliato al numero di caratteri nella stringa specificata. La stringa di caratteri deve essere racchiusa tra parentesi solo se contiene dei separatori.
Per esempio:

.NCHR SYM,MESSAGGIO

SYM è uguagliato a 9 (decimale).
Questa direttiva può essere usata in un punto qualsiasi di un programma. Con la direttiva .NTYPE si può determinare il modo di indirizzamento dei parametri della macro che si sta espandendo.
Il formato è:

.NTYPE simbolo,parametro

Il simbolo è uguagliato al valore dei 6 bits che danno il modo di indirizzamento del parametro (formale).
La direttiva .NTYPE può essere usata solo all'interno di una definizione di macro.

19.4. .ERROR, .PRINT

La direttiva .ERROR è usata per far stampare un messaggio durante il secondo passo dell'assemblaggio. Il formato è:

.ERROR expr;testo

dove:

expr è una espressione il cui valore viene stampato quando si incontra la direttiva .ERROR; se l'espressione non è specificata, viene stampato solo il testo

testo è la stringa che viene stampata

; indica l'inizio della stringa

Quando incontra una direttiva .ERROR l'Assembler stampa una riga contenente le seguenti informazioni:

1) numero di sequenza della direttiva .ERROR
2) valore corrente del location counter
3) valore dell'espressione, se specificata
4) la stringa specificata.

La linea è segnalata nella lista assembly con un codice errore P. La direttiva .PRINT è analoga a .ERROR, soltanto che non viene segnalato il codice di errore P.

19.5. .IRP, .IRPC

Le direttive .IRP e .IRPC servono per creare delle ripetizioni indefinite di blocchi di istruzioni. Una ripetizione indefinita è essenzialmente una definizione di macro che ha soltanto un parametro formale ed è espansa una volta per ciascun parametro reale. Una ripetizione indefinita è in linea con la sua definizione, invece che essere richiamata tramite il nome, come le macro.
Una ripetizione indefinita può essere inserita all'interno o all'esterno di una definizione di macro, di un rango di ripetizioni, di un rango di ripetizioni indefinite. Per creare un blocco di ripetizioni indefinite si usano le stesse regole viste per la definizione di macro.
La frase iniziale è

.IRP arg,<parametri attuali>

dove:

arg è un parametro formale che è successivamente sostituito con i parametri attuali, nella frase .IRP .

I parametri attuali, che devono essere racchiusi in parentesi acute, sono liste di zero o più caratteri o a loro volta una lista di parametri attuali racchiusi in parentesi acute.
I parametri attuali sono separati da virgola.

Lo statement .IRP può essere etichettato a meno che non sia all'interno di un'altra definizione di macro o di una ripetizione.
Il rango delle ripetizioni può contenere anche definizioni di macro, ed altri blocchi di ripetizioni.
Un blocco di ripetizioni è chiuso dalla direttiva .ENDM

Esempio

```
.IRP X,<A,B,C>
MOV X,(RØ)+
.ENDM
```

genera il codice:

```
MOV A,(RØ)+
MOV B,(RØ)+
MOV C,(RØ)+
```

La direttiva .IRPC è usata in modo analogo. Il formato è:

```
.IRPC arg,stringa
```

Ad ogni iterazione, però, il parametro formale (arg) viene sostituito con un carattere della stringa.

Esempio

```
.IRPC X,ABCD
.ASCII /X/
.ENDM
```

genera il codice:

```
.ASCII /A/
.ASCII /B/
.ASCII /C/
.ASCII /D/
```

19.6. .REPT

Può essere talora utile duplicare lo stesso blocco di codice più volte in linea col programma sorgente. Ciò si ottiene creando un blocco di ripetizioni nel modo seguente:

```
 .REPT  espressione
 :
(blocco di istruzioni da ripetere)
 :
 .ENDM  (o .ENDR)
```

Il blocco di codice è ripetuto un numero di volte pari al valore dell'espressione. Il blocco di codice può contenere definizioni di macro, blocchi di ripetizioni indefinite, altri blocchi di ripetizioni, condizionali. La frase .REPT può essere etichettata a meno che non si trovi all'interno di una definizione di macro, di un altro blocco di ripetizioni, o di un blocco di ripetizioni indefinite. All'interno di un blocco di ripetizioni nessuna frase può essere etichettata.
La frase finale di un blocco di ripetizioni deve essere una direttiva .ENDM o .ENDR, e non può essere etichettata.

Esempio

```
.REPT  1Ø
.WORD  Ø
.ENDM
```

genera il codice:

```
.WORD  Ø
.WORD  Ø
.WORD  Ø
.WORD  Ø
.WORD  Ø
.WORD  Ø
.WORD  Ø
.WORD  Ø
```

19.7. Le librerie macro

Ogni macro deve essere definita prima di essere richiamata all'interno di un programma. Se un programma usa delle macro di sistema, il programmatore deve indicare quali definizioni di macro sono richieste.
Si usa, quindi, la direttiva .MCALL per specificare i nomi di tutte le macro non definite nel programma corrente ma richieste dal programma stesso. La direttiva .MCALL deve apparire prima della prima chiamata di una macro esterna. Il formato di questa direttiva è:

```
.MCALL nome1,nome2,...
```

dove nome1,nome2,... sono i nomi delle macro richieste nel programma corrente.
Quando incontra la direttiva .MCALL, il Macro Assembler ricerca nella libreria di sistema SYSMAC.SML le definizioni richieste.

20. LA PROGRAMMAZIONE IN CODICE INDIPENDENTE DALLA POSIZIONE

L'uscita di un assemblaggio MACRO è un modulo oggetto, che sarà unito dal LINK ad altri eventuali moduli per creare un modulo eseguibile. Una volta costruito, un programma può essere caricato ed eseguito solo all'indirizzo specificato al momento del collegamento; questo perché alcuni codici sono modificati dal LINK relativamente alle locazioni di memoria in cui il programma deve essere eseguito.
E' possibile, però, scrivere un programma che può essere caricato ed eseguito in ogni sezione di memoria. Un tale programma consiste di codice indipendente dalla posizione (PIC-position independent code). Il fatto che un codice risulti o meno indipendente dalla posizione di memoria in cui è caricato, dipende dai modi di indirizzamento usati.
Tutti i modi di indirizzamento che usano soltanto i registri generali, cioè:

```
  R
@R
  (R)+   @(R)+
 -(R)    @-(R)
```

sono indipendenti dalla posizione, se, come è lecito, si suppone che il contenuto dei registri non dipenda da una particolare posizione di memoria.
I modi di indirizzamento relativo,

A e @A

sono generalmente indipendenti dalla posizione.
Non risultano indipendenti dalla posizione quando A è un indirizzo assoluto riferito da una sezione rilocabile. I modi indice sono indipendenti dalla posizione se la base, per calcolare l'in

dirizzo, è indipendente dalla posizione. Per esempio:

```
        MOV  2(SP),RØ
```

è indipendente dalla posizione, così come:

```
        N = 5
        MOV N(SP),RØ
```

mentre:

```
    ADDR:
        CLR  ADDR(R1)
```

non è indipendente dalla posizione, perché la base, ADDR, dipende dalla posizione di memoria in cui il programma è caricato.
Il modo di indirizzamento immediato,#N,è indipendente dalla posizione solo se N è un valore assoluto, e non, per esempio, una etichetta.
Il modo di indirizzamento assoluto,@#A, usato per ottenere il contenuto di un indirizzo specificato, è chiaramente dipendente dalla posizione. E' indipendente dalla posizione solo se A è un indirizzo assoluto.
Il codice indipendente dalla posizione è usato per scrivere programmi come ad esempio i programmi per la gestione dei periferici, e le routine di sistema, che possono essere caricate in ogni parte di memoria.
Nella lista di un programma in assembly,il MACRO segnala con un apice,', il contenuto di ogni parola che richiede un intervento del LINK. In alcuni casi l'apice segnala che il codice è dipendente dalla posizione, in altri serve per richiamare l'attenzione dell'utente sull'uso di un simbolo che può non essere indipendente dalla posizione.

Per scrivere programmi in codice indipendente dalla posizione bisogna, anche, tener presente la distinzione fra espressioni assolute, rilocabili e esterne.

a. Una espressione è *assoluta* se il suo valore è fissato, per esempio una espressione i cui termini sono numeri o conversioni ASCII.

b. Una espressione è *rilocabile* se il suo valore è fissato relativamente ad un indirizzo di base, ma alla quale il LINK dovrà aggiungere un valore; per esempio le espressioni i cui termini contengono etichette rilocabili sono rilocabili.

c. Una espressione è *esterna* (o globale) se il suo valore è soltanto parzialmente definito durante l'assemblaggio ed è completato al momento del collegamento. Per esempio le espressioni i cui termini contengono un simbolo globale non definito nel programma corrente, sono esterne. Le espressioni esterne hanno un valore rilocabile al momento della esecuzione se il simbolo globale è stato definito come rilocabile, o assoluto se il simbolo globale è definito come assoluto.

PARTE TERZA
IL SISTEMA MONITOR

In questa parte vengono descritte le richieste programmate che eseguono le operazioni di I/O nel sistema operativo DOS BATCH-Monitor.

21. INTRODUZIONE

Il Disk-Operating System (DOS) Monitor è un sistema disegnato per l'uso sugli elaboratori PDP-11.
Il sistema DOS-Monitor è di supporto all'utente PDP-11 nella stesura ed esecuzione del programma perché:
a) procura un accesso ai programmi di sistema, come il compilatore FORTRAN, l'assemblatore MACRO-11, Link, ecc.
b) esegue i trasferimenti di ingresso/uscita a tre livelli distinti.
c) procura un sistema di files per la gestione della memoria secondaria.
d) procura un insieme di comandi da tastiera (keyboard) per controllare il flusso dei programmi.

Le operazioni di ingresso/uscita (I/O) manipolate dal Monitor, sono possibili a tre livelli detti READ/WRITE, RECORD/BLOCK e TRAN.
Un'operazione di I/O a livello READ/WRITE è un trasferimento con formato in cui l'utente può scegliere fra nove opzioni possibili.
A livello RECORD/BLOCK l'operazione di I/O è ad accesso casuale su una struttura di file, senza formato.
A livello TRAN, infine, l'ingresso/uscita è eseguito con operazioni a livello delle routines di sistema per la gestione dei periferici ("driver").
Il sistema di files sulla memoria secondaria usa due tipi di files: *contigui* e *concatenati* (linked). La lunghezza dei files contigui deve essere dichiarata prima dell'uso ma su questi files si possono fare operazioni di I/O ad accesso casuale, cioè a livello RECORD/BLOCK. I files concatenati possono essere estesi e non

hanno limite logico alla loro grandezza. I blocchi in un file contiguo sono fisicamente adiacenti, mentre in un file concatenato sono tipicamente non adiacenti (la prima voce di ciascun blocco contiene l'indirizzo del blocco successivo). I files possono essere creati e cancellati in ogni momento, e sono riferiti per nome.
L'utente può comunicare con il Monitor in due modi: per mezzo di istruzioni da keyboard, dette *comandi*, e per mezzo di istruzioni programmate dette *richieste*.
Con i comandi da keyboard l'utente può caricare e mandare in esecuzione un programma, assegnare mezzi di I/O o files, modificare il contenuto delle posizioni di memoria, richiedere certe informazioni al sistema, come per esempio la data e l'ora.
Le richieste programmate sono macro, assemblate nel programma utente, per mezzo delle quali il programmatore specifica le operazioni che il Monitor deve eseguire.
Se le richieste programmate sono usate per eseguire delle operazioni di I/O, il Monitor si prende cura di portare in memoria, dal disco, i drivers necessari , di eseguire il trasferimento di dati e comunicare all'utente lo stato del trasferimento.
Altre richieste programmate possono essere usate per richiedere certe informazioni al sistema, come la data e l'ora, lo stato del sistema e per specificare funzioni speciali per il mezzo periferico.
La memoria centrale è divisa in:

a) un'area utente, dove sono caricati i programmi utenti;

b) lo stack, dove vengono trasferiti temporaneamente i parametri durante il trasferimento di controllo fra i programmi;

c) un'area libera o area di buffer che è divisa in blocchi di 16 parole assegnati dal Monitor per le tabelle temporanee per le routines del Monitor caricate dal disco e per la memorizzazione transitoria di dati fra i periferici e i programmi utenti;

d) il Monitor residente, che include tutte le routines e tavole permanenti;

e) i vettori di interruzione (interrupt) (cfr. parte IV).

Una tabella è un insieme di dati memorizzati in posizioni di memoria sequenziali. Per esempio:

TAV :			
	A		
	C	B	(1)
	D		

Fig. 16 - Esempio di tabella

è una tabella di 3 parole che è riferita con l'indirizzo simbolico TAV. Il primo ingresso è all'indirizzo TAV e contiene A, per cui può essere codificata nel programma utente con

```
.WORD A
```

La seconda parola della tavola, all'indirizzo TAV + 2 è divisa in due bytes; il byte di destra contiene B mentre quello di sinistra contiene C; infine la terza voce, all'indirizzo TAV + 4, contiene D. La rappresentazione della tabella (1) in un programma è quindi:

```
TAV : .WORD A
      .BYTE B,C
      .WORD C
```

Nel seguito sono illustrate le principali richieste programmate, rimandando alle consultazioni del manuale [2] per l'uso dei comandi e delle altre richieste.

22. LE RICHIESTE PROGRAMMATE

Come abbiamo detto, l'utente può comunicare con il Monitor per mezzo di richieste programmate.
Una richiesta programmata è, essenzialmente, una chiamata di macro, eventualmente seguita da uno o più parametri, che è assemblata nel programma utente e interpretata dal Monitor al momento dell'esecuzione.
Al momento dell'assemblaggio il MACRO Assembler espande la macro in una sequenza di istruzioni che passano il controllo e i parametri alla appropriata routine del Monitor per eseguire la funzione specificata.
Poiché si tratta di macro di sistema, i nomi delle macro dovranno comparire in una direttiva .MCALL prima del loro uso nel programma.
Per esempio:

```
.INIT #LNKBLK        (1)
```

è una richiesta programmata seguita dal parametro #LNKBLK. L'espansione di questa macro è:

```
MOV #LNKBLK,-(SP)
EMT 6
```

La macro .INIT deve comparire come parametro in una direttiva .MCALL, ossia prima della frase (1) deve esserci:

```
.MCALL .INIT
```

In questo modo si avverte l'Assembler che il programma richiede la definizione della macro .INIT .
Le macro di sistema accettano nel trasferimento di parametri quasi tutti i modi di indirizzamento. Per esempio, poiché il Monitor per eseguire la richiesta .INIT, (cfr.§24.1) si aspetta in testa

allo stack l'indirizzo di una tabella,(Link block), ciascuna delle seguenti chiamate della macro può essere appropriata:

```
.INIT #LNKBLK
.INIT RØ      ; l'indirizzo LNKBLK è in RØ
.INIT PUNT    ; l'indirizzo LNKBLK è in PUNT
```

Gli argomenti di una richiesta sono parametri o indirizzi di tabelle(del tipo descritto precedentemente) che contengono i parametri. In tal caso, le tabelle sono parte del programma utente. Esclusa una piccola porzione permamentemente residente, le routines del Monitor che eseguono le richieste programmate sono sul disco e vengono caricate, dal Monitor, in memoria soltanto quando è necessario. L'utente può, tuttavia, specificare che una o più di queste routines trasportabili sia residente in memoria centrale in modo permanente o solo per la durata del suo programma. Facendo diventare residente una certa routine si ha una maggiore occupazione di memoria, ma si ha pure un risparmio di tempo nella esecuzione della richiesta associata.
Ogni routine che serve una richiesta programmata può diventare residente in uno dei modi seguenti:

a) in modo permanente al momento della generazione del Monitor;
b) solo per la durata del programma utente se il suo nome globale compare come parametro in una direttiva assembler .GLOBL all'interno del programma stesso.

Le operazioni rese possibili dal Monitor per mezzo delle richieste programmate possono essere divise in tre gruppi:

1) operazioni di ingresso/uscita
2) gestione di files
3) altre operazioni, come conversioni di dati, richiesta di parametri del Monitor, ecc.

Si fa inoltre presente che, se non altrimenti specificato, quando una richiesta è completata il controllo ritorna alla istruzione che segue l'espansione in linguaggio assemblativo e i parametri sono rimossi dallo stack.

23. LE OPERAZIONI DI I/O

Le operazioni di I/O sono possibili a tre livelli di trasferimento di dati:

i) READ o WRITE
ii) RECORD o BLOCK
iii) TRAN

Ciascun livello di trasferimento è realizzato con una sequenza di richieste. E' da tener presente la distinzione fra READ/WRITE, RECORD/BLOCK e TRAN come nomi di livelli di trasferimento e .READ, .WRITE, .RECORD, .BLOCK e .TRAN come richieste programmate all'interno di questi livelli.

24. IL LIVELLO READ/WRITE

La maggior parte delle operazioni di I/O si effettua a questo livello. I dati sono manipolati in ordine sequenziale, nel senso che ciascuna lettura o scrittura è applicata al record (o linea) successivo nel file. I records possono avere lunghezza variabile e possono avere formato ASCII o binario.
Le operazioni di I/O a questo livello consistono in un trasferimento di dati fra un periferico e un buffer dell'utente, interno al programma, attraverso un buffer del Monitor.

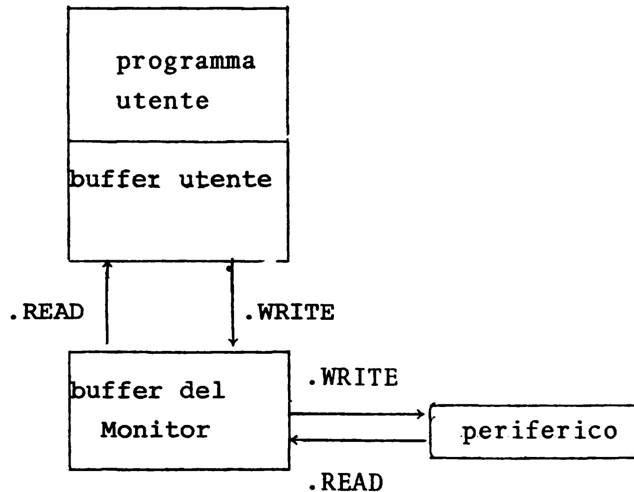


Fig. 17 - Trasferimento dei dati a livello READ/WRITE

Il buffer in linea è un'area di memoria stabilita dall'utente nel quale egli (o il Monitor) pone i dati per la scrittura (o lettura).
Il buffer utente è generalmente preceduto da una *testata* (line buffer header) in cui l'utente specifica la grandezza e la posizione del buffer e il formato dei dati.
Per realizzare un trasferimento a questo livello, l'utente deve prima di tutto associare il mezzo di I/O con un file (dataset) per mezzo della richiesta .INIT, con la quale si assicura anche che il driver relativo sia caricato in memoria. Dopo la richiesta .INIT, il programmatore apre il dataset con la richiesta .OPENx .
Questa richiesta è però necessaria solo per i mezzi (come ad esempio il disco), su cui è possibile memorizzare i dati per nome, piuttosto che semplicemente per locazioni fisiche (file-structured device). Un dataset può essere aperto per l'ingresso, l'uscita, per l'aggiornamento o l'estenzione. L'ultima lettera della richiesta .OPENx specifica quale tipo di apertura è richiesto. Seguono poi .READ e .WRITE rispettivamente per la lettura e scrittura.

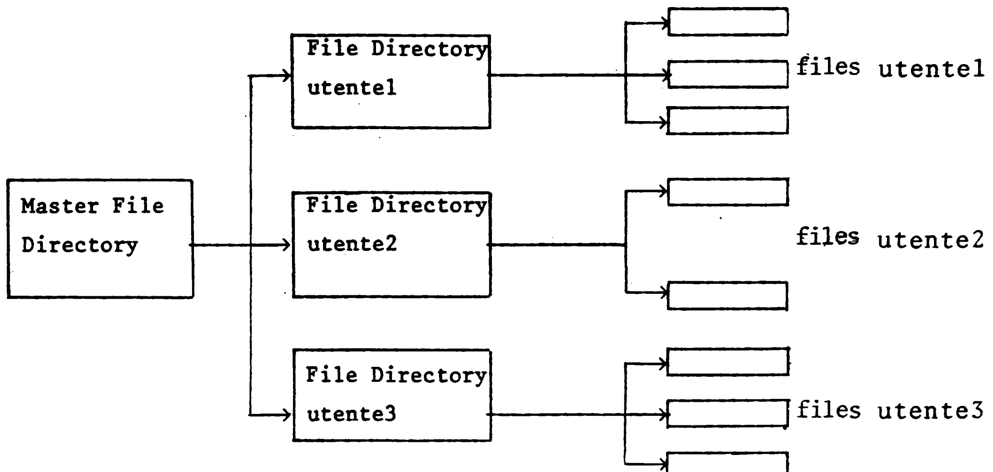
Per controllare se l'ultimo trasferimento è stato completato si usa la richiesta .WAIT, che in caso affermativo passa il controllo all'istruzione successiva.
Se tutti i dati sono stati trasferiti, con la richiesta .CLOSE si completano tutte le azioni pendenti, si aggiornano tutti i directories(1) interessati e si libera la memoria che il Monitor aveva allocato per il buffer. Infine, l'operazione sul dataset termina con la richiesta .RLSE che disassocia il periferico dal dataset, e rilascia il driver, liberando così la memoria.

24.1. .INIT

Chiamata della macro:

```
.INIT #LNKBLK
```

(1) Un "File-Directory" dell'utente (UFD) è una lista dei files che contiene per ciascun file il nome, la posizione e il codice di protezione, in altre parole ogni informazione che riguarda il file ma non è parte di esso. Ciascun UFD è associato con un codice di identificazione dell'utente (UIC). Le posizioni degli UFDs sul mezzo e i nomi degli UICs con cui sono associati, sono contenuti in un "Master File Directory".



dove LNKBLK è l'indirizzo del così detto Link Block , cioè della tabella che contiene le informazioni necessarie.
L'espansione in linguaggio assemblativo è:

```
MOV #LNKBLK,-(SP)
EMT 6  (1)
```

Il nome globale è: INR .
Questa richiesta associa un periferico con un dataset e assicura che il driver relativo al mezzo sia in memoria. Infatti se il driver non è in memoria, viene caricato.
Il mezzo assegnato è quello specificato nel Link Block, a meno che non sia stato fatto un assegnamento diverso al nome logico del dataset con un comando ASSIGN.
Il formato del Link Block è:

	Indirizzo di ritorno in caso di errori	
LNKBLK :	ØØØØØØ (puntatore)	
	nome logico del dataset	
	n.ro dell'unità	n.ro di parole che seguono
	nome del periferico in formato (Radix 5Ø)	

Fig. 18 - Formato del Link Blook

LNKBLK-2 indirizzo di ritorno questo punto d'ingresso della tabella deve essere specificato dall'utente; contiene l'indirizzo a cui si vuol passare

(1) Per l'istruzione EMT cfr. §39

		il controllo nel caso in cui si verifichi qualche errore in una richiesta associata con questo dataset. Se non è specificato nessun indirizzo, in caso di errore l'esecuzione viene interrotta;
LNKBLK	puntatore	questa posizione deve essere messa a zero dall'utente e non deve essere modificata. Viene usata solo dal Monitor che vi pone un indirizzo di collegamento quando il dataset è inizializzato;
LNKBLK-2	nome logico	in questa parola l'utente può specificare un nome per il dataset. Questo è memorizzato nella forma Radix-5Ø con la direttiva assembler .RAD5Ø;
LNKBLK+4	n.ro di parole	questo byte contiene il numero di parole che seguono nel blocco; sarà Ø se l'utente non specifica il nome fisico del mezzo, altrimenti è 1;
LNKBLK+5	n.ro dell'unità	questo byte specifica il numero dell'unità del mezzo associato al dataset. Per esempio se si tratta del disco Ø (DKØ) o del disco 1 (DK1);
LNKBLK+6	nome fisico del mezzo	specifica il nome standard del mezzo nel formato Radix-5Ø. Se non viene specificato

		nessun nome, l'utente deve indi care un nome logico per il data set e dare un comando ASSIGN prima di mandare in esecuzione il programma;
LNKBLK+8 fino a LNKBLk+n	dati opzionali	queste parole sono presenti so lo se il byte all'indirizzo LNKBLK+4 è maggiore di 1. Sono usate per passare informazioni aggiuntive.

Ogni dataset che è stato inizializzato, deve essere dissociato dal mezzo (con la richiesta .RLSE) prima di essere associato ad un altro mezzo.

```
        .INIT #INPUT
          .
          .
          .
          .
 ; LINK BLOCK
        .WORD 0                  ; INDIRIZZO DI RITORNO
 INPUT : .WORD 0                 ; PUNTATORE
        .RAD50 /INP/             ; NOME LOGICO DEL DATASET
        .BYTE 1                  ; NUMERO DI PAROLE CHE SEGUONO
        .BYTE 0                  ; NUMERO DELL' UNITA'
        .RAD50 /B1/              ; NOME DEL PERIFERICO
          .
          .
```

Fig. 19 - Richiesta .INIT

24.2. .OPEN

La richiesta .OPEN prepara un periferico per il trasferimento di dati e associa il dataset con un file, (se il mezzo e file-structured). Una forma di chiamata è:

```
.OPEN #LNKBLK,#FILBLK
```

dove LNKBLK è l'indirizzo del Link Block (già inizializzato) e FILBLK è l'indirizzo di una tabella all'interno del programma utente detta File Block , che in questo caso contiene anche il tipo di apertura. L'espansione in linguaggio assemblativo è:

```
MOV #FILBLK,-(SP)
MOV #LNKBLK,-(SP)
EMT 16
```

L'altra forma di chiamata della macro è:

```
.OPENx#LNKBLK,Rn
```

dove Rn è un registro che contiene l'indirizzo del File Block e x indica il tipo di apertura. L'espansione in assembly è:

```
MOV #CODE,-2(Rn)
MOV Rn,-(SP)
MOV #LNKBLK,-(SP)
EMT 16
```

Il nome globale della richiesta è: OPN.
Se è usata, la richiesta .OPEN segue .INIT o .CLOSE se più di un file deve essere aperto sullo stesso dataset. Se il mezzo usato sopporta strutture di file, .OPEN associa uno specifico file con il dataset. La richiesta .OPEN ha 5 forme, che possono essere

specificate inserendo un opportuno codice di apertura nel File Block oppure scegliendo una delle forme alternative di chiamata della macro. Le forme possibili sono:

forma	codice di apertura	descrizione
.OPENU	1	apre un file contiguo, creato precedentemente, per operazioni di I/O con richieste .RECRD e .BLOCK (cfr.§25.1 e §26.1);è usabile solo se il mezzo è file-structured ;
.OPENO	2	i) crea un nuovo file concatenato e lo prepara per l'uscita con la richiesta .WRITE; ii) prepara un mezzo non file-structured per l'uscita con .WRITE ;
.OPENE	3	apre un file contiguo o concatenato, precedentemente creato per farlo più lungo per mezzo della richiesta .WRITE; è da tener presente che un file contiguo può essere esteso solo all'interno dell'area allocata, mentre ad un file concatenato si possono aggiungere nuovi blocchi ;
.OPENI	4	i) apre un file contiguo o concatenato, già creato per l'ingresso tramite .READ, .RECRD, .BLOCK; ii) prepara un mezzo non file-structured per l'ingresso tramite .READ ;

.OPENC 13 apre un file contiguo, già creato, per l'uscita tramite .WRITE; quando si apre per la prima volta un file contiguo per l'uscita, tramite .WRITE, si deve usare .OPENC; per le successive aperture per scrittura, con .WRITE, si deve usare .OPENE .

Le richieste di trasferimento possibili per ciascun tipo di apertura possono essere riassunte nella tabella seguente:

	Files concatenati		Files contigui				
	ingresso	uscita	ingresso		uscita		
Tipo di aperura	.READ	.WRITE	.READ	.RECRD .BLOCK	.WRITE	.RECRD .BLOCK	file esiste?
.OPENU				SI		SI	SI
.OPENO		SI					NO
.OPENE		SI			SI		SI
.OPENI	SI		SI	SI			SI
.OPENC					SI		SI

Fig. 20 - Le richieste possibili per ciascun tipo di apertura di un file.

La richiesta .OPENO non è applicabile a files contigui: crea un file concatenato, per cui il file non deve esistere prima della richiesta .OPENO .
Se un file è aperto per la lettura (input) (OPENI) non può essere aperto per l'uscita, ma può essere aperto per l'estensione o aggiornamento.
Se il file è aperto con .OPENU, .OPENE, .OPENC deve esistere e non può essere aperto da un altro .OPENU, .OPENE o .OPENC.

Il formato del File Block è il seguente:

FILBLK :

indirizzo di ritorno	
codice di errore	codice di apertura
nome del file	
nome del file	
estensione	
codice di identificazione	
vuoto	codice di prot.

Fig. 21 - Formato del File Block

FILBLK-4	indirizzo di ritorno	si può specificare in questa parola l'indirizzo cui si vuole che il Monitor passi il controllo nel caso in cui si verifichi un errore. Se non è specificato nessun indirizzo, in caso di errore viene interrotta l'esecuzione;
FILBLK-2	codice di apertura	in questo byte si specifica il codice di apertura secondo lo schema visto in precedenza;
FILBLK-1	codice di errore	questo codice è posto dal Monitor per indicare il tipo di errore che si è verificato. Per sapere a quale tipo di errore corrisponde ciascun codice si rimanda alla consultazione del manuale [2];
FILBLK FILBLK+2	nome del file	in queste due parole l'utente specifica il nome del file nel formato Radix-5Ø. Il nome del file non è necessario se il dataset non è un file.

FILBLK+4	estensione	in questa parola si può specificare,in formato Radix-50, l'eventuale estensione. Il nome di un file è costituito da un massimo di 6 caratteri alfanumerici, mentre l'estensione da un massimo di 3 ed è separata con un punto (.) dal nome del file. Per esempio in ELIS.OBJ ELIS è il nome del file e OBJ è l'estensione;
FILBLK+6	codice di identificazione dell'utente	l'utente può porre in questa parola il proprio codice di identificazione, che è costituito da una coppia: mmm,nnn dove mmm (nnn) rappresenta una stringa da 2 a 3 cifre ottali. Sarà nnn nel byte di destra ed mmm nel byte di sinistra. Se non è specificato nessun codice di identificazione, viene assunto quello corrente;
FILBLK+10	codice di protezione	l'utente può qui specificare la protezione da dare al file. Se è 0 viene assunto il codice default 233.

Nel codice di identificazione le cifre mmm rappresentano il numero del gruppo, mentre le cifre nnn rappresentano il numero dell'utente all'interno del gruppo.
Nel codice di protezione ciascun sottoinsieme di 3 bits proibisce ad una certa classe di utenti un certo tipo di accesso. Ad un file si può accedere per la lettura, scrittura, cancellazione e per mandarlo in esecuzione. Gli utenti possono essere classificati in 3 classi, relativamente al file in oggetto:

a) l'utente stesso, cioè colui che ha definito il file;
b) gli utenti che appartengono allo stesso gruppo;
c) tutti gli altri utenti.

Il byte che definisce il codice di protezione è così suddiviso

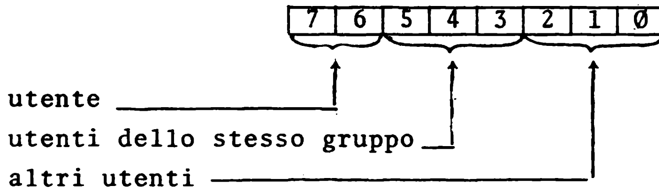


Fig.22-Formato del codice di protezione di un file.

bit 7 = 1 protegge il file dalla cancellazione automatica al comando FINISH

bit 6 = 1 l'utente non può scrivere o cancellare il file.

Per gli altri due gruppi le operazioni possibili per ogni codice sono illustrate nella tabella seguente, dove "SI" indica che l'operazione è permessa.

codice	cancell.	scritt.	lett.	esec.
Ø	SI	SI	SI	SI
1		SI	SI	SI
2 o 3			SI	SI
4 o 5				SI
6 o 7				

Fig.23-Operazioni possibili per ciascun codice di protezione.

Per esempio se un file appartiene all'utente [2Ø,2Ø] e il codice di protezione è 3 l'utente [18,18] può leggere o mandare in esecuzione il file ma non può scrivere o cancellare il file.

```
        .OPEN #INPUT ,#FILE
          .
          .
          .
          .
; FILE BLOCK
        .WORD ERR                   ; INDIRIZZO DI RITORNO
        .BYTE 4                     ; APERTURA PER INPUT
        .BYTE 0                     ; ERRORI
FILE:   .RAD50 /ELI/                ;NOME DEL FILE
        .RAD50 /SA/
        .RAD50 /INP/                ; ESTENSIONE
        .WORD 0                     ; CODICE DI IDENTIFICAZIONE
        .BYTE 0                     ; CODICE DI PROTEZIONE
        .EVEN
          .
          .
          .
```

Fig.24 - Richiesta .OPEN

24.3. .READ e .WRITE

La chiamata della macro .READ è:

.READ #LNKBLK,#BUFHDR

dove LNKBLK è l'indirizzo del Link Block e BUFHDR è l'indirizzo della testata del buffer utente, cioè di una tabella che contiene le informazioni necessarie per il trasferimento.
L'espansione in linguaggio assemblativo è:

```
MOV #BUFHDR,-(SP)
MOV #LNKBLK,-(SP)
EMT 4
```

Il nome globale è RWN, ma la routine è residente.
La richiesta .READ trasferisce i dati dal mezzo al buffer del-

l'utente, tramite un buffer nel Monitor in cui viene trasferito un intero blocco del mezzo. In ciascuna operazione viene letto il record successivo nel dataset. Se il mezzo è file-structured la richiesta .READ deve essere preceduta dalla richiesta .OPENI Dopo che ogni operazione di I/O è iniziata il controllo ritorna all'utente all' istruzione successiva e gli argomenti sono tolti dallo stack. Sebbene ogni altra azione sul dataset da parte del Monitor sia postposta finché l'operazione di lettura (o scrittura) non sia terminata, è opportuno far seguire la richiesta .READ (o .WRITE) da una richiesta .WAIT o .WAITR, per assicurarsi che il trasferimento sia effettivamente completato prima di manipolare i dati.

La richiesta .WRITE scrive il record successivo nel dataset.
La chiamata è:

```
.WRITE #LNKBLK,#BUFHDR
```

dove LNKBLK è l'indirizzo del Link Block e BUFHDR è l'indirizzo della testata del buffer utente.
L'espansione in assembly è:

```
MOV #BUFHDR,-(SP)
MOV #LNKBLK,-(SP)
EMT 2
```

Il nome globale è RWN, ma la routine è residente.
I dati sono prima trasferiti in un buffer del Monitor, dove sono accumulati fino a riempire un buffer di una determinata lunghezza che dipende dal periferico usato. A questo punto i dati dal buffer del Monitor sono trasferiti nel blocco successivo del periferico e gli eventuali dati rimasti nel buffer dell'utente vengono messi nel buffer del Monitor.
Se il mezzo usato è file-structured il dataset deve essere stato precedentemente aperto con una richiesta .OPENO o .OPENE per

un file concatenato, oppure .OPENC per un file contiguo.
Su un dataset non si possono fare contemporaneamente una operazione di lettura e una di scrittura; pertanto se è necessario usare lo stesso mezzo per entrambi le operazioni si devono usare due datasets diversi.
Come abbiamo visto, sia per le operazioni di lettura che per le operazioni di scrittura, l'utente deve stabilire un buffer e la sua testata nel programma. Il formato della tabella che definisce la testata è:

BUFHDR:

n.ro massimo di bytes	
stato	modo
n.ro attuale di bytes	
puntatore (solo per il modo indiretto)	

Fig.25-Formato della testata del buffer utente

BUFHDR	n.ro massimo di bytes	questo numero indica la grandezza del buffer in bytes.Deve essere specificato dall'utente in ogni operazione di lettura ;
BUFHDR+2	modo	l'utente specifica qui il modo di trasferimento ;
BUFHDR+3	stato	il Monitor pone in questo byte lo stato del trasferimento quando il controllo ritorna all'utente. In questo byte vengono segnalati anche gli eventuali errori;
BUHHDR+4	n.ro attuale di bytes	questo numero deve essere inizializzato dall'utente e indica il numero di bytes da trasferire in una operazione di scrittura ;

BUFHDR+6 puntatore se il bit 2 nel byte che definisce il modo di trasferimento è 1, l'utente deve specificare in questa voce l'indirizzo di inizio del buffer utente. Se il bit 2 del modo è Ø la testata è composta solo da tre parole e il buffer deve seguire fisicamente la testata.

Il byte in cui è specificato il modo di trasferimento ha il seguente formato:

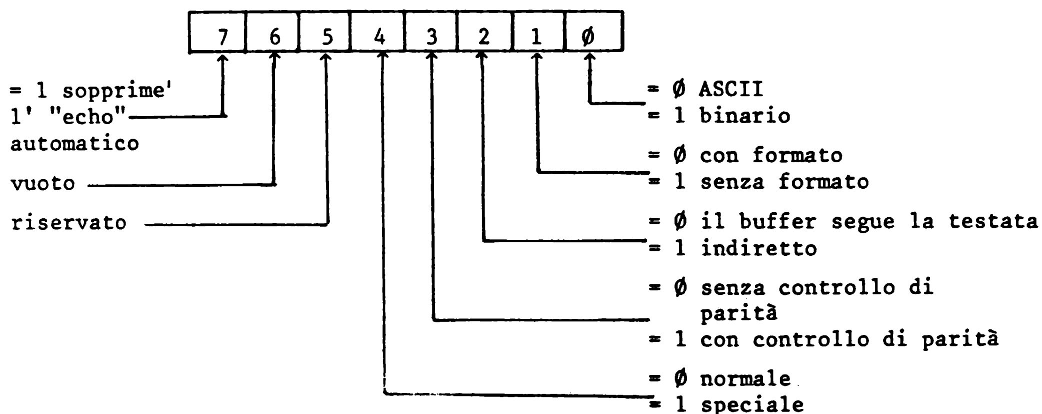


Fig.26-Formato del byte contenente il modo di trasferimento

---

(1) Se viene usata come mezzo di ingresso dei dati la keyboard, i dati trasmessi vengono riscritti sul video.
Se il bit 7 è = 1 i caratteri trasmessi non appariranno sul video.

I dati possono essere binari o ASCII e si possono avere nove tipi di trasferimento.

per i dati ASCII: con formato e con controllo di parità e speciale
" " " " " " " normale
" " senza " " " " speciale
" " " " " " " normale
senza formato con controllo di parità e speciale
senza formato senza controllo di parità e normale

per i dati binari: con formato - speciale
con formato - normale
senza formato - normale

1) Con formato ASCII normale

I dati sono assunti dal Monitor come insieme di caratteri ASCII in 7 bits che terminano con controlli di fine riga, o fine pagina o tabulatori (caratteri terminali).

lettura: i dati sono trasferiti nel buffer utente finché viene incontrato un carattere terminale o il numero dei bytes trasferiti raggiunge il massimo (indicato in BUFHDR); se il massimo è raggiunto prima di incontrare un carattere terminale, i caratteri rimanenti, fino al carattere terminale, sono sovrapposti nell'ultimo byte del buffer e viene segnalato un errore nel byte di stato. Dopo il trasferimento il numero attuale di bytes è uguagliato al numero di bytes letti, compresi quelli in eccesso;

scrittura: i dati vengono trasferiti dal buffer utente finché il numero di bytes trasferiti non uguaglia il numero attuale di bytes. Se l'ultimo carattere trasferito non è terminale, viene messo a 1 il bit nel bytes di stato che segnala il verificarsi di un errore in una li

nea. I precedenti caratteri terminali sono trasferiti come normali caratteri.

2) Con formato ASCII speciale

lettura: in questo caso se il numero massimo di bytes è raggiunto prima di un carattere terminale il trasferimento è interrotto e i dati in eccesso non sono sovrapposti ma vengono mantenuti per la successiva .READ. Il numero attuale di bytes è uguagliato al numero massimo;

scrittura: i dati sono trasferiti finché non viene incontrato il primo carattere terminale; se il numero attuale di bytes viene raggiunto prima di aver incontrato un carattere terminale il trasferimento è interrotto e viene segnalato un errore nel byte di stato.
In questo modo si può, quindi trasferire, soltanto una riga alla volta, ma non è necessario che il numero attuale di bytes sia esatto, purché sia maggiore del numero effettivo dei dati da trasferire.

3) Con formato binario normale

scrittura: è un trasferimento di 8 bits; vengono trasferite anche le parole 2 e 3 (Stato/modo e numero attuale di bytes) della testata del buffer e i dati sono trasferiti finché il numero di caratteri trasferiti non è uguale al numero attuale di bytes; poi è calcolata la somma di caratteri trasferiti, che viene scritta alla fine; tale somma riflette la presenza delle parole 2 e 3 della testata.

lettura: il buffer è riempito finché non viene raggiunto il numero attuale di bytes o il numero massimo. Se il massimo è raggiunto prima del numero attuale viene segnalato un errore nel byte di stato e i caratteri rimanenti sono sovrapposti nell'ultimo byte del buffer. Dopo il trasferimento il numero attuale di bytes è uguagliato al numero di bytes letti.

4) Con formato binario speciale

scrittura: identica al modo normale;

lettura: se il numero massimo è raggiunto prima dell'attuale i dati non sono sovrapposti. Il numero attuale di bytes è uguagliato al numero di dati che dovevano essere letti e quindi potrà essere fatto un controllo da parte dell'utente per verificare se effettivamente sono stati letti tutti i caratteri del record.

5) Modo senza formato ASCII speciale e normale

Vengono trasferiti 7 bits, l'ottavo è posto uguale a zero. I caratteri nulli sono scartati.

lettura: il trasferimento viene interrotto quando si raggiunge il numero massimo di bytes

scrittura: i dati sono trasferiti finché non viene raggiunto il numero attuale di bytes

6) Modo senza formato binario normale e speciale

Questo modo è identico al modo senza formato ASCII con la differenza che vengono trasferiti 8 bits e non vengono scartati i caratteri nulli. Non viene calcolata la somma dei caratteri trasferiti.

7) Modo con formato ASCII con controllo di parità

Identico al modo con formato ASCII (speciale o normale), eccetto che viene controllato se il numero di bits= 1 è pari. In uscita viene posto a 1 l'ottavo bit, se il numero di bits= 1 è dispari altrimenti viene posto a Ø. In ingresso viene controllata la parità. I caratteri corretti vengono trasmessi all'utente in 7 bits mentre quelli non corretti vengono segnalati ponendo a 1 il bits, 8 e il bit 5 del byte di stato.

8) Modo senza formato ASCII con controllo di parità

Identico al modo senza formato ASCII, eccetto che vengono trasferiti 8 bits. Non viene né generata né controllata la parità.

9) Modi indiretti

Il modo indiretto significa che il buffer non segue la testata e che la quarta voce della testata deve essere interpretata come puntatore all'inizio del buffer. Questi modi sono detti anche *DUMP* .

Il formato del byte di stato è:

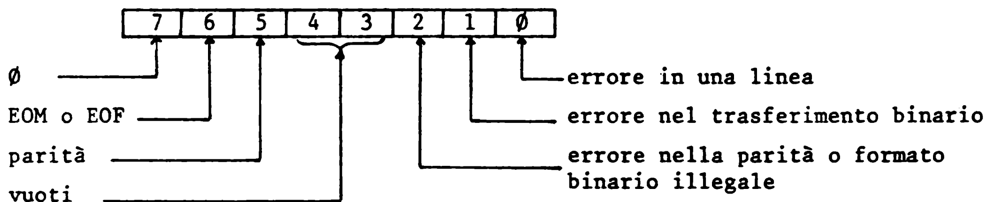


Fig.27-Formato del byte di stato

Per maggiori informazioni sulla funzione di ciascun bit si rimanda al manuale [2]

```
        .INIT #INPUT
          .
          .
          .
        .READ #INPUT,#LEGGI       ; LETTURA
        .WAIT #INPUT              ; ATTESA
          .
          .
          .
        .INIT #OUT
        .WRITE #OUT,#SCRIVI
        .WAIT #OUT
          .
          .
          .
; TESTATA DEL BUFFER
LEGGI:  .WORD 81.                 ; NUMERO MASSIMO DI BYTES
        .BYTE 0                   ; MODO DI TRASFERIMENTO
        .BYTE 0                   ; STATO
        .WORD 81.                 ; NUMERO ATTUALE DI BYTES
; BUFFER IN LINEA
BUFF:   .BLKB 81.
        .EVEN
          .
          .
          .
SCRIVI: .WORD 81.                 ; NUMERO MASSIMO DI BYTES
        .BYTE 4                   ; MODO DI TRASFERIMENTO
        .BYTE 0                   ; STATO
        .WORD 0
        .WORD 81.                 ; NUMERO ATTUALE DI BYTES
        .WORD BUFF                ; PUNTATORE AL BUFFER
        .EVEN
          .
          .
        .WORD 0
OUT:    .WORD 0
        .RAD50 /OUT/
        .WORD 1
        .RAD50 /LP/
          .
          .
```

Fig. 28 - Richieste .READ e .WRITE

24.4. .WAIT e .WAITR

La chiamata della macro .WAIT è:

```
.WAIT #LNKBLK
```

dove LNKBLK è l'indirizzo del Link Block. L'espansione in linguaggio assemblativo è:

```
MOV #LNKBLK,-(SP)
EMT 1
```

La routine fa parte del Monitor residente.
Controlla se è stata completata l'ultima azione richiesta sul dataset rappresentato nel Link Block. Se l'azione è completata il controllo ritorna all'utente all' istruzione che segue l'espansione in assembly ; altrimenti il controllo resta al Monitor finché l'azione non sia completata.
La richiesta .WAITR è analoga a .WAIT, soltanto si può specificare un indirizzo a cui viene passato il controllo nel caso in cui l'azione non sia completata. La chiamata della macro è:

```
.WAIT #LNKBLK,#ADDR
```

dove ADDR è, appunto, l'indirizzo, specificato dall'utente, a cui verrà passato il controllo se l'azione sul dataset non è stata completata. L'espansione in linguaggio assemblativo è:

```
MOV #ADDR,-(SP)
MOV #LNKBLK,-(SP)
EMT Ø
```

(Anche questa routine fa parte del Monitor residente).
Sarà quindi compito dell'utente ritornare all' istruzione .WAITR

per ulteriori controlli.

24.5. .CLOSE

La chiamata della macro è:

.CLOSE #LNKBLK

l'espansione in linguaggio assemblativo è:

```
MOV #LNKBLK,-(SP)
EMT 17
```

Il nome globale è CLS.
Questa richiesta indica che il Monitor non deve eseguire nessun altra operazione di I/O sul dataset indicato nel Link Block. La richiesta .CLOSE completa ogni operazione pendente, per esempio in uscita scrive l'ultimo buffer, in una operazione di estensione, collega l'estensione al file precedente, e aggiorna i directories interessati. Quando un file aperto per l'uscita è chiuso l'ultimo blocco e l'ultimo byte scritto sono segnalati nel directory per indicare la fine dei dati.

24.6. .RLSE

La chiamata della macro è:

.RLSE #LNKBLK

dove LNKBLK è l'indirizzo del Link Block precedentemente inizializzato.
L'espansione in linguaggio assemblativo è:

```
MOV #LNKBLK,-(SP)
EMT 7
```

Il nome globale è: RLS.
Con questa richiesta il dataset viene dissociato dal mezzo e il driver relativo viene tolto dalla memoria.
Se il dataset era stato aperto, prima di essere dissociato dal mezzo deve essere chiuso.

```
        .CLOSE #INPUT
          .
          .
          .
        .RLSE #INPUT
          .
          .
        .INIT #OUT
          .
          .
          .
        .WORD 0
OUT:    .WORD 0
        .RAD50 /OUT/
        .WORD 1
        .RAD50 /LP/
          .
          .
```

Fig.29-Richieste .CLOSE e .RLSE

25. IL LIVELLO RECORD

Le operazioni di I/O a livello RECORD sono usate per un accesso casuale ai records in un file. Le richieste a livello RECORD sono applicabili solo a mezzi file-structured e a files contigui. I records del file devono avere tutti la stessa lunghezza, cioè lo stesso numero di bytes; non è necessario formattare i records o usare caratteri terminali.
Il modo più comune per creare un file su cui si eseguono operazioni a questo livello consiste nell'aprire il file con .OPENC (dopo averlo allocato) e scrivere con .WRITE usando i modi di trasferimento senza formato (ASCII o binario). Quando il file è

chiuso la fine logica del file viene fissata dopo l'ultimo record scritto. Successive letture del file con .READ o .RECRD dovranno essere limitate all'area scritta. Successivamente il file può essere aperto per l'estensione (.OPENE) e nuovi dati possono essere scritti (.WRITE).
Un altro modo per creare un file di questo tipo consiste nell'aprire il file con .OPENU e scrivere i dati con .RECRD.
Il questo modo la fine logica del file corrisponde alla fine fisica dell'area allocata.
Le operazioni di I/O a questo livello consistono in un trasferimento di dati fra il periferico ed un buffer interno al programma utente.

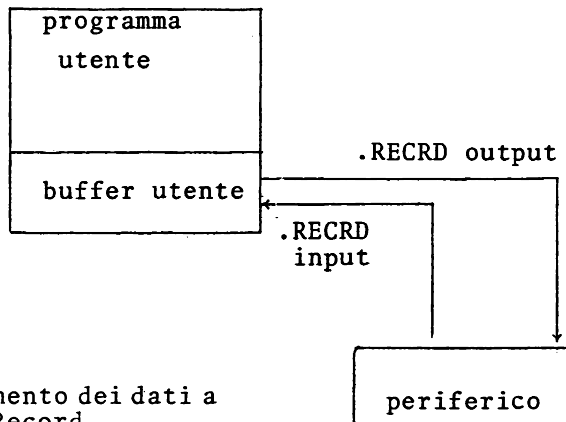


Fig.30-Trasferimento dei dati a livello Record.

La richiesta che realizza i trasferimenti, nei due sensi, a questo livello è .RECRD .
Tale richiesta deve essere preceduta da .INIT (il mezzo associato deve essere file-structured), da .OPENx, e seguita da .WAIT o .WAITR . Quando il procedimento sul file è terminato si dovrà usare la richiesta .CLOSE; e infine la richiesta .RLSE indicherà che le operazioni sul dataset sono terminate.

25.1. .RECRD

La chiamata della macro è:

```
.RECRD #LNKBLK,#RECBLK
```

dove LNKBLK è l'indirizzo del Link Block e RECBLK è l'indirizzo di una tabella interna al programma detta Record Block, che contiene l'indirizzo del buffer, il numero del record e la direzione del trasferimento. L'espansione in linguaggio assemblativo è:

```
MOV #RECBLK,-(SP)
MOV #LNKBLK,-(SP)
EMT 25
```

Il nome globale è REC.
Esegue il trasferimento in (o da) un buffer utente di uno specifico record, non necessariamente il successivo. Il trasferimento avviene attraverso un buffer nel Monitor che contiene esattamente un blocco fisico. Non esiste alcuna regola riguardo alla grandezza deirecords relativamente alla grandezza dei blocchi,ma perefficienza è conveniente che uno sia multiplo dell'altro.
Il file associato alla richiesta .RECRD deve essere stato aperto con .OPENU oppure .OPENI se si devono fare solo operazioni di lettura.
Il formato del Record Block è il seguente:

RECBLK:	
	funzione / stato
	indirizzo del buffer
	lunghezza del record
	numero del record

Fig.31-Formato del Record Block

RECBLK funzione/stato.

I bits= 1 in questa voce hanno il seguente significato:

bit

Ø - non usato
1 - record output - stabilito dall'utente
2 - record input - " " "
3-8 - non usati.
(il valore dei bits da 9 a 15 è stabilito dal Monitor).
9 - funzione illegale
1Ø - file non contiguo oppure il mezzo non è file-structured
11 - il record richiesto è al di fuori del file
12 - file non aperto
13 - codice di protezione violato, apertura non corretta
14 - non usato
15 - errore nel periferico.

L'utente deve stabilire il valore dei bits 1 e 2 mentre gli altri bits sono stabiliti dal Monitor, ed è conveniente che siano controllati dall'utente dopo ogni richiesta .RECRD per assicurarsi che la richiesta sia completata correttamente.

RECBLK+2	indirizzo del buffer	questa parola contiene l'indirizzo del buffer. Questo buffer deve essere sufficientemente grande da contenere un intero record, della lunghezza indicata nella voce seguente
RECBLK+4	lunghezza del record	questa parola contiene la lunghezza del record espressa in bytes.E' bene ricordare che tutti i records del file devono avere la stessa lunghezza.

RECBLK+6 RECBLK+10	numero del record	queste parole identificano il record in oggetto, ossia contengono il numero del record da trasferire. Il primo record ha il numero Ø. Sono previste due parole, in quanto un file può contenere anche più di 65535 records.

```
        .INIT #DISK
        MOV #FILBLK,R0
        .OPENU #DISK,R0
        .RECRD #DISK,#REC1
        .WAIT #DISK
          .
          .
          .
REC1:   .WORD 4                   ; INPUT
        .WORD BUFF                ; PUNTATORE
        .WORD 20                  ; LUNGHEZZA DEL RECORD
        .WORD 0,2                 ; RECORD NUMERO DUE
          .
          .
          .
        .WORD 0
DISK:   .WORD 0
        .RAD50 /FIL/
        .WORD 1
        .RAD50 /DK/
          .
          .
        .WORD 0
        .WORD 0
FILBLK: .RAD50 /PRO/
        .RAD50 /VA/
        .RAD50 /DAT/
        .BYTE 200,200
        .WORD 0
          .
          .
```

Fig.32-Sequenza di richiesta per il livello RECORD

26. IL LIVELLO BLOCK

Il livello BLOCK è analogo al livello RECORD, soltanto ad ogni operazione di I/O si ha il trasferimento di un intero blocco fisico di dati invece che di una quantità di dati definita dall'utente. Le richieste a questo livello possono essere usate solo con mezzi con directory (cioè disco, ma non nastro magnetico) e con file contigui. Inoltre il trasferimento dei dati avviene da o in un buffer stabilito dal Monitor piuttosto che in un buffer stabilito dall'utente.

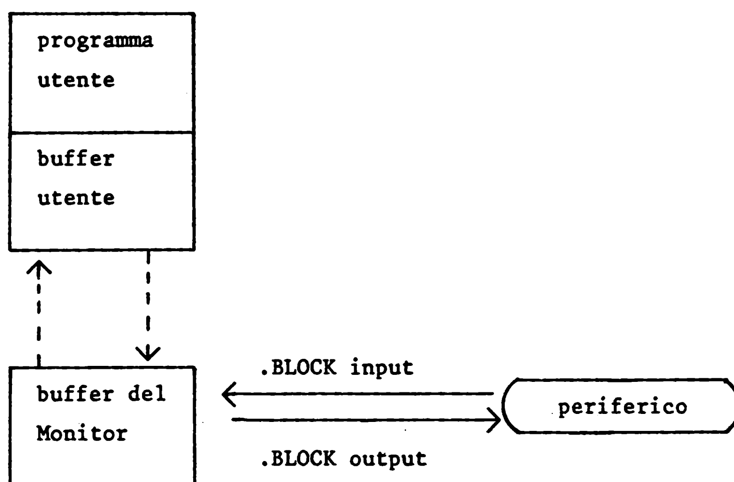


Fig.33-Trasferimento dei dati a livello BLOCK

L'utente può trasferire, a programma, i dati in un buffer interno al suo programma oppure può elaborarli direttamente nel buffer del Monitor.
La richiesta programmata che realizza i trasferimenti a questo livello è .BLOCK, che deve essere preceduta da .INIT e .OPEN e seguita da .WAIT, .CLOSE.

26.1. .BLOCK

La richiesta .BLOCK procura un accesso casuale ai blocchi di un file memorizzato su disco, o sull'unità a nastro DECtape
La chiamata della macro è:

```
.BLOCK #LNKBLK,#BLKBLK
```

dove LNKBLK è l'indirizzo del Link Block e BLKBLK è l'indirizzo di una tabella del programma che contiene le informazioni riguardanti il trasferimento, detta Block Block.
L'espansione in linguaggio assemblativo è:

```
MOV #BLKBLK,-(SP)
MOV #LNKBLK,-(SP)
EMT 11
```

Il nome globale è BLO.
I dati sono trasferiti a o da uno specificato blocco in un file, senza formato. Il trasferimento avviene fra un blocco sul mezzo e un buffer nel Monitor. Il file associato deve essere stato aperto con .OPENI per l'input o con .OPENU per input o output.
Si ricorda poi, che il file deve essere contiguo.
L'utente deve specificare una delle tre funzioni: INPUT,GET, OUTPUT.

INPUT : durante una richiesta INPUT, viene letto nel buffer del Monitor il blocco richiesto, e nel Block Block viene messo l'indirizzo del buffer e la lunghezza fisica del blocco trasferito ;

GET : durante una richiesta GET il Monitor pone nel Block Block l'indirizzo e la lunghezza di un buffer al suo interno che l'utente potrà usare per successive operazioni di scrittura. Questo buffer potrà essere usato

per più operazioni. La richiesta GET non è necessaria se è già stata eseguita una richiesta INPUT, in quanto è già stato allocato un buffer all'interno del Monitor;

OUTPUT : durante una richiesta OUTPUT il contenuto del buffer assegnato è scritto sul mezzo nel file e nella posizione richiesta.

Il formato del Block Block è il seguente:

BLKBLK:
funzione/stato
numero del blocco
indirizzo del buffer
lunghezza

Fig.34-Formato del Block Block

BLKBLK funzione/stato l'utente specifica la funzione da eseguire, e il Monitor ritorna con lo stato appropriato.

bit	bit = 1 significa:
Ø	la funzione è GET
1	la funzione è INPUT
2	la funzione è OUTPUT
3-8	riservati
9	funzione illegale
1Ø	il file non è contiguo e il mezzo non è adatto
11	il numero del blocco non esiste nel file
12	file non aperto
13	violazione del codice di protezione
14	errore nella fine dei dati

		15 errore nel mezzo.
BLKBLK+2	n.ro del blocco	in questa parola deve essere posto il numero del blocco da trasferire, relativamente all'inizio del file. Il primo blocco del file ha il numero Ø
BLKBLK+4	indirizzo del buffer	il Monitor pone in questa parola lo indirizzo del buffer, nelle funzioni INPUT e GET
BLKBLK+6	lunghezza	il Monitor pone in questa parola la lunghezza del buffer in parole.

```
        MOV #FILBLK,R0
        .OPENU #DISK,R0
        .BLOCK #DISK,#BLOC
        .WAIT #DISK
          .
          .
          .
        .WORD 0
DISK:   .WORD 0
        .RAD50 /FIL/
        .WORD 1
        .RAD50 /DK/
        .WORD 0
        .WORD 0
FILBLK: .RAD50 /PROVA/
        .RAD50 /DAT/
        .BYTE 200,200
        .WORD 0
          .
          .
          .
BLOC:   .WORD 4                 ; INPUT
        .WORD 0                 ; NUMERO DEL BLOCCO
        .WORD 0                 ; QUESTA VOCE CONTERRA' L' INDIRIZZO
                                ; DEL BUFFER NEL MONITOR
        .WORD 0                 ; QUESTA VOCE CONTERRA' LA LUNGHEZZA
                                ; DEL BLOCCO
          .
          .
```

Fig.35-Sequenza di richieste per il livello BLOCK

27. IL LIVELLO TRAN

Una richiesta a livello TRAN è una operazione base di I/O. La richiesta dell'utente è semplicemente passata al driver appropriato. La richiesta programmata .TRAN che realizza le operazioni a questo livello non riconosce alcuna struttura di file, per cui tale richiesta va usata con molta cautela e possibilmente solo quando è strettamente necessario in quanto si potrebbero facilmente danneggiare in modo irreparabile le informazioni sul mezzo. I dati sono trasferiti direttamente fra il mezzo e un buffer stabilito dall'utente.

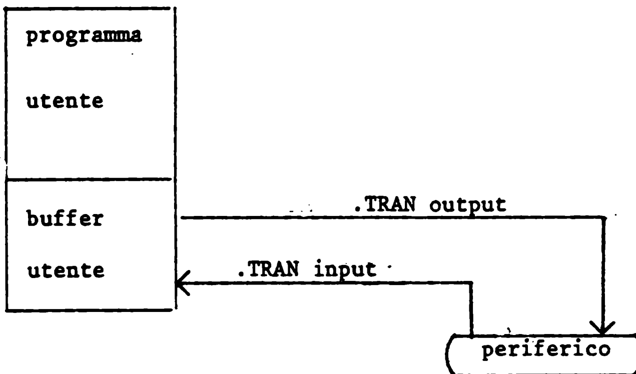


Fig.36-Trasferimento dei dati a livello TRAN

Per implementare una richiesta a livello TRAN il programmatore deve usare .INIT e .RLSE ma *non* deve usare .OPEN e .CLOSE.

27.1. .TRAN

La chiamata della macro è:

```
.TRAN #LNKBLK, #TRNBLK
```

dove LNKBLK è l'indirizzo del Link Block e TRNBLK è l'indirizzo del così detto Tran Block, cioè della tabella che contiene le informazioni necessarie al trasferimento.
L'espansione in linguaggio assemblativo è:

```
MOV #TRNBLK,-(SP)
MOV #LNKBLK,-(SP)
EMT 1Ø
```

Il nome globale è TRA.
La richiesta .TRAN procura un accesso quasi diretto al mezzo su cui il dataset risiede.Ciascuna richiesta .TRAN trasferisce uno o più blocchi, senza considerare alcuna struttura di file. I blocchi sui mezzi file-structured sono riferiti per numero assoluto di blocco, mentre su gli altri mezzi sono trasferiti in ordine sequenziale.
La richiesta .TRAN deve essere preceduta da .INIT ma non deve essere usata la richiesta .OPEN. Per assicurarsi che il trasferimento sia completo è opportuno usare .WAIT (o .WAITR).
Nel Tran Block è specificato l'indirizzo di memoria da cui inizia il buffer utente, l'indirizzo del blocco nel mezzo, il numero di parole da trasferire e la funzione da eseguire. .TRAN è perciò una richiesta che dipende dal mezzo.
Il formato del TRAN block è:

TRNBLK :	
	numero del blocco nel mezzo
	indirizzo iniziale di memoria
	contatore di parole
	funzione/stato
	numero di parole non trasferite

Fig.37-Formato del Tran Block

TRNBLK numero del blocco
l'utente deve specificare in questa parola il numero assoluto del blocco del mezzo, se questo è file-structured, da cui il trasferimento deve iniziare. Il primo blocco ha il numero Ø. Se il mezzo non è file-structured, in questa parola deve essere mezzo Ø;

TRNBLK+2 indirizzo del buffer
l'utente deve specificare l'indirizzo di memoria centrale dal quale inizia il trasferimento di dati;

TRNBLK+4 contatore di parole
l'utente specifica in questa parola il numero totale di parole a 16 bits che devono essere trasferite.

TRNBLK+6 funzione/stato

bit	significato del bit
Ø	= 1 binario = Ø ASCII
1	= 1 scrittura
2	= 1 lettura
3-1Ø	riservati per uso del Monitor-11
11	direzione del DECtape Ø = avanti 1 = indietro
12	riservato per il sistema operativo RSX-11
13	chiamata sbagliata
14	fine del mezzo (EOM)
15	errore nel mezzo (ad es. parità o lunghezza del record)

I bits 13,14,15 sono usati dal Monitor;

TRNBLK+1Ø numero di parole non trasferite
questa voce deve essere lasciata vuota dall'utente. Se avviene una EOM durante il trasferimento, il Monitor porrà in questa voce il numero di parole non trasferite.

```
        .INIT #DISK
          .
          .
          .
        .TRAN #DISK,#TR
          .
          .
          .
        .RLSE #DISK
          .
          .
          .
        .WORD 0
          - D 0
        .WORD 1
        .RAD 50 /DK/
          .
          .
          .
TR:     .WORD 200              ; NUMERO DEL BLOCCO
        .WORD BUFF             ; INDIRIZZO DEL BUFFER
        .WORD 200              ; NUMERO DI PAROLE
        .WORD 4                ; INPUT
        .WORD 0
          .
          .
          .
BUFF:   .BLKW 200
          .
          .
          .
```

Fig.38-Richiesta .TRAN

28. .SPEC

Per specificare alcune funzioni particolari per un periferico, come per esempio riavvolgere un nastro magnetico, si usa la richiesta .SPEC..La chiamata di questa macro è:

.SPEC #LNKBLK,#SPCARG

dove LNKBLK è l'indirizzo del Link Block e SPCARG è un codice oppure l'indirizzo di una tabella che contiene il codice relativo alla funzione richiesta.
L'espansione in linguaggio assemblativo è:

```
MOV #SPCARG,-(SP)
MOV #LNKBLK,-(SP)
EMT 12
```

Il nome globale è SPC.
Se la funzione richiesta non richiede alcun dato di supporto il parametro della macro è il codice stesso. Se invece sono necessarie altre informazioni, oltre il codice di identificazione della funzione, oppure se la funzione restituisce dei dati all'utente il parametro della macro è l'indirizzo di una tabella che, fra l'altro, contiene il codice della funzione.
Il formato di questa tabella è:

SPCBLK :

n.parole che seguono	codice
altre informazioni	

Fig.39-Formato della tabella per la richiesta .SPEC

SPCBLK	codice	questo byte contiene il codice di identificazione della funzione che è un numero compreso tra Ø e $255_{1Ø}$;
SPCBLK+1	n.ro parole	l'utente indica qui quante altre parole appartengono a questo blocco. La grandezza di un blocco dipende, infatti, dalla funzione richiesta;

SPCBLK+2 e segg.	in queste voci l'utente porrà le eventuali informazioni richieste dalla funzione oppure saranno messi i risultati della funzione stessa.

I codici delle funzioni speciali sono:

codice	funzione
1	spento (offline)
2	scrive la fine del file ("end of file")
3	riavvolge (rewind)
4	salta il record (o i records) in avanti
5	salta il record (o i records) indietro
6	stabilisce la densità o la parità
7	ottiene lo stato
8	stabilisce la grandezza del buffer
9	attiva o disattiva la funzione di riavvolgimento
1Ø	salta i files in avanti
11	salta i files indietro
12	fa un' operazione di READ dopo una verifica di scrittura.

I codici 1Ø-11-12 sono usati solo per le cassette.
In generale i codici delle funzioni speciali hanno significati analoghi da periferico a periferico.

28.1. Le funzioni speciali per il nastro magnetico

Il driver del nastro magnetico richiede un blocco speciale per eseguire la richiesta di funzioni particolari.
Dovrà quindi essere usata la richiesta .SPEC nel formato:

```
.SPEC #LNKBLK,#SFBLK
```

Il formato del blocco SFBLK è:

SFBLK :

numero di parole	codice
stato dell'unità a nastro	
contatore o informazione di controllo	
contatore residuo	

Fig.40-Formato del blocco per il nastro magnetico

SFBLK: codice — codice della funzione. Le funzioni possibili per il nastro magnetico sono quelle identificate dai codici 1-9;

SFBLK+1 numero parole — questo byte contiene il numero di parole che seguono. Per il nastro magnetico il numero di parole deve essere maggiore o uguale a 3;

SFBLK+2 stato — in questa parola il driver pone lo stato dell'unità a nastro, dopo la richiesta della funzione con codice 7;

SFBLK+4 contatore — in questa parola l'utente può porre il numero di records da saltare per le funzioni con codice 4 o 5;

SFBLK+6 contatore residuo — in questa voce il driver pone il numero di records residui, cioè dei records che non ha potuto saltare perché ha incontrato un "end-of-file" (EOF) o il marcatore di inizio del nastro (BOT).

La funzione identificata dal codice 4 fa scorrere il nastro di tanti records quanti sono specificati in SFBLK+4. Se prima di aver saltato tutti i records richiesti incontra un record EOF, questo record è contato, ma l'operazione termina e il numero degli e-

ventuali records non saltati viene messo in SFBLK+6. La funzione identificata dal codice 5 è analoga alla precedente; se viene incontrato il marcatore BOT prima di aver saltato tutti i records richiesti, questo non viene contato ma l'operazione termina, e il numero dei records residui viene posto in SFBLK+6.
La richiesta con codice 1 fa riavvolgere il nastro e mette in OFF lo stato di SELECT REMOTE . L'operazione con codice 3 è analoga a questa, nel senso che riavvolge il nastro ma non mette in OFF lo stato di SELECT REMOTE .
E' opportuno che queste due operazioni siano precedute da una richiesta .CLOSE durante le operazioni di READ/WRITE perché potrebbero causare la perdita di informazioni in quanto, se l'ultimo comando per il driver era WRITE, prima di riavvolgere il nastro vengono scritti tre "EOF".

29. .STAT

Questa richiesta può essere usata per conoscere le caratteristiche di un particolare periferico. La chiamata della macro è:

```
.STAT #LNKBLK
```

dove LNKBLK è l'indirizzo del Link Block.
L'espansione in linguaggio assemblativo è:

```
MOV #LNKBLK,-(SP)
EMT 13
```

Il nome globale è: STT.
Il periferico interessato è quello specificato nel Link Block.
Questa richiesta restituisce le seguenti informazioni in testa allo stack:

SP parola che illustra le caratteristiche del periferico

SP+2 nome del periferico nel formato Radix-50

SP+4 grandezza (in parole) di un buffer del periferico.

Il nome del periferico è nella forma simbolica, (per esempio CR per il lettore di schede e LP per la stampante, MT per il nastro magnetico).
La grandezza standard del buffer è la grandezza di un blocco, misurata in parole, se il mezzo è a blocchi, oppure di un appropriato raggruppamento sui periferici a caratteri (come ad es. la stampante).
La parola che illustra le caratteristiche del periferico ha il seguente formato:

bit	bit = 1 significa
0	il periferico sopporta l'attività di datasets multipli
1	" " gestisce l'uscita
2	" " " l'ingresso
3	" " " dati binari
4	" " " dati ASCII
5	il driver ha una speciale funzione di ingresso
6	" " ha una entrata CLOSE
7	" " " " " OPEN
8	il periferico è un terminale
9	" " è un nastro sequenziale a cassetta
10	" " ha più unità
11	" " sopporta lunghezze multiple di records
12	" " è un disco
13	" " è un nastro magnetico
14	" " è file-structured

Fig.41-Formato della parola che contiene le caratteristiche di un periferico dopo l'esecuzione della richiesta .STAT

Esempio

Per ottenere queste informazioni relative al nastro magnetico si può usare il seguente programma:

```
        .TITLE PROVA STAT
        .MCALL .INIT,.STAT,.BIN2O,.RADUP,.BIN2D,.RLSE
        .MCALL .WRITE,.WAIT,.EXIT
UNO:
        .INIT #IN
        .STAT #IN
        .RLSE #IN
        .INIT #OUT
        MOV (SP)+,WORD
        MOV (SP)+,NAME
        MOV (SP)+,BUFF
        .BIN2O #OTW,WORD
        .RADUP #ASNA,NAME
        .BIN2D #BLOCK,BUFF
        .WRITE #OUT,#STATO
        .WAIT #OUT
FINE:
        .RLSE #OUT
        .EXIT
; DEFINIZIONE DEI FILES DI I/O
        .WORD 0
IN:     .WORD 0
        .RAD50 /ING/
        .WORD 1
        .RAD50 /MI/
        .WORD 0
OUT:    .WORD 0
        .RAD50 /STA/
        .WORD 1
        .RAD50 /LP/
STATO: .WORD 100.,0,<B-A>
A:      .BYTE 14,12,15
        .ASCII /            ** DRIVER FACILITIES WORD **/
        .BYTE 12,15,40,40,40,40,40,40,40,40,40,40
OTW:    .BLKB 6
        .BYTE 12,15
        .ASCII /            ** DEVICE NAME **/
        .BYTE 12,15,40,40,40,40,40,40,40,40,40,40
ASNA:  .BLKB 3
        .BYTE 12,15
        .ASCII /            ** DEVICE STANDARD BUFFER SIZE **/
        .BYTE 12,15,40,40,40,40,40,40,40,40,40,40
BLOCK: .BLKB 5
        .BYTE 14
B:      .BYTE 12
        .EVEN
NAME:  .BLKW 1
WORD:  .BLKW 1
BUFF:  .BLKW 1
       .END UNO
```

che produce l'uscita seguente:

```
** DRIVER FACILITIES WORD **
024177
** DEVICE NAME **
MT
** DEVICE STANDARD BUFFER SIZE **
00256
```

30. LE RICHIESTE PER LA GESTIONE DEI FILES

Le richieste programmate illustrate in questo paragrafo sono usate per definire files, ricercare i files nei directories, aggiornare i nomi dei files e i codici di protezione.

30.1. .ALLOC

La richiesta .ALLOC è usata per allocare, cioè creare, un file contiguo. La chiamata della macro è:

.ALLOC #LNKBLK,#FILBLK,#N

dove LNKBLK è l'indirizzo del Link Block, FILBLK è l'indirizzo del File Block e N è il numero di segmenti di 64 parole richieste.

L'espansione in linguaggio assemblativo è:

```
MOV #N,-(SP)    oppure MOV #N+100000,-(SP)
MOV #FILBLK,-(SP)
MOV #LNKBLK,(-SP)
EMT 15
```

Il nome globale è ALO.

Viene ricercata un'area libera di N segmenti di 64 parole e viene creato un file contiguo in questa area, se è trovata, ponendo un appropriato punto di ingresso nel directory dell'utente. Se il bit del segno di N è 1, il puntatore nel directory punterà all'inizio dell'area allocata indicando quindi che il file è vuoto. In questo caso sarà quindi possibile riempire parzialmente il file e successivamente estenderlo. Altrimenti il puntatore nel directory punterà alla fine dell'area allocata, indicando quindi che l'area del file è piena e non può essere successivamente estesa.
E' opportuno ricordare che i files concatenati sono creati con la richiesta .OPENO .
Il numero di blocchi allocati è il numero minimo necessario per contenere N segmenti di 64 parole.
Per esempio sul disco i blocchi sono composti da 256_{10} parole; quindi ci sono 4 segmenti di 64 parole per ogni blocco.
Pertanto se N = 10_{10} verranno allocati 3 blocchi.
La richiesta .ALLOC deve essere preceduta da .INIT .
Dopo che la richiesta è stata completata il controllo ritorna all'utente all'istruzione che segue l'espansione in linguaggio assemblativo. I parametri sono rimossi dallo stack, e in testa allo stack viene posto -1 se la richiesta termina con successo, oppure il numero massimo di segmenti accessibili se questo è minore di quello richiesto.
In caso di errore il controllo ritorna all'indirizzo di errore nel File Block, se è specificato; altrimenti viene mandato un messaggio di errore sul video.

Es.

Definire un file contiguo sul disco (DK) zero che occupi 2 blocchi di 256_{10} parole. Il nome del file è PROVA .DAT.

```
        .ALLOC #DISK,#FILBLK ,#10
          .
          .
          .
        .WORD 0
DISK:   .WORD 0
        .RAD50 /FIL/
        .WORD 1
        .RAD50 /DK/
        .WORD 0
        .WORD 0
FILBLK: .RAD50 /PRO/
        .RAD50 /VA/
        .RAD50 /DAT/
        .BYTE 200,200
        .WORD 0
```

30.2. .DELET

La richiesta .DELET è usata per cancellare un file. La chiamata della macro è:

```
.DELET #LNKBLK,#FILBLK
```

dove LNKBLK è l'indirizzo del LNKBLK e FILBLK è l'indirizzo del File Block. L'espansione in linguaggio assemblativo è:

```
MOV #FILBLK,-(SP)
MOV #LNKBLK,-(SP)
EMT 21
```

Il nome globale è DEL.
Questa richiesta è usata per cancellare dal directory il file il cui nome è indicato nel File Block.
La richiesta .DELET può essere usata per cancellare sia files contigui che files concatenati. Se il file è stato aperto, prima di essere cancellato deve essere chiuso.
In caso di errore la richiesta .DELET si comporta come la richiesta .ALLOC. .

30.3. .RENAM

La richiesta .RENAM è usata per cambiare il nome e il codice di protezione di un file. La chiamata della macro è:

```
.RENAM #LNKBLK,#OLDNAM,#NEWNAM
```

la cui espansione in linguaggio assemblativo è:

```
MOV #NEWNAM,-(SP)
MOV #OLDNAL,-(SP)
MOV #LNKBLK,-(SP)
EMT 2Ø
```

Il nome globale della macro è REN.
Con questa richiesta l'utente può cambiare il nome e il codice di protezione del file. Il dataset deve essere inizializzato e il file non deve essere aperto.
Il blocco all'indirizzo OLDNAM contiene il nome e il codice di protezione del file che vogliamo cambiare; il blocco all'indirizzo NEWNAM contiene il nuovo nome e il nuovo codice di protezione, cioè il nome e il codice di protezione che il file deve

avere dopo la richiesta .RENAM .I due nomi dei files devono essere diversi, per cui per cambiare solo il codice di protezione sono necessarie due richieste .RENAM. .
Non deve esistere un altro file con lo stesso nome.
In caso di errore la richiesta .RENAM si comporta come la richiesta .ALLOC. .

30.4. .APPND

La richiesta .APPND serve per unire un file concatenato ad un altro file.
La chiamata è:

```
.APPND #LNKBLK,#FIRST,#SECOND
```

dove LNKBLK è l'indirizzo del LNKBLK, FIRST è l'indirizzo della tavola che contiene le informazioni relative al primo file, cioè al file cui verrà aggiunto il file descritto nella tavola all'indirizzo SECOND.
L'espansione in linguaggio assemblativo è:

```
MOV #SECOND,-(SP)
MOV #FIRST;-(SP)
MOV #LNKBLK,-(SP)
EMT 22
```

e il nome globale è APP.
Con questa richiesta si crea un file concatenato aggiungendo il file descritto nella tavola all'indirizzo SECOND al file descritto in FIRST.
Il punto di ingresso del file aggiunto è tolto dal directory.
I due file non sono compattati, cioè i blocchi fisici sono semplicemente collegati.
Poiché l'ultimo blocco di un file non è generalmente riempito

del tutto, ci saranno dei caratteri nulli nel nuovo file nel punto di congiunzione. Questo fatto non causa problemi particolari per i files di tipo ASCII, ma va tenuto ben presente per i files di tipo binario.
In caso di errore anche questa richiesta si comporta come .ALLOC.

30.5. .KEEP

Questa richiesta serve per proteggere il file dalla cancellazione automatica. La chiamata della macro è:

```
.KEEP #LNKBLK,#FILBLK
```

dove LNKBLK è l'indirizzo del Link Block e FILBLK è l'indirizzo della tavola che contiene le informazioni sul file da proteggere.
Con questa richiesta il file viene protetto dalla cancellazione automatica in seguito al comando FINISH. Ciò è ottenuto ponendo a 1 il bit 7 nel byte che contiene il codice di protezione nella tavola FILBLK.
L'espansione in linguaggio assemblativo è:

```
MOV #FILBLK,-(SP)
MOV #LNKBLK,-(SP)
EMT 24
```

Il nome globale è PRO.

30.6. .LOOK

La richiesta .LOOK serve per ricercare un file. Questa macro può avere due o tre parametri, ossia la chiamata può essere:

```
.LOOK #LNKBLK,#FILBLK
```

oppure:

```
.LOOK #LNKBLK,#FILBLK,1
```

Nel primo caso l'espansione in linguaggio assemblativo è:

```
MOV #FILBLK,-(SP)
MOV #LNKBLK,-(SP)
EMT 14
```

mentre nel secondo caso è:

```
MOV #FILBLK,-(SP)
CLR -(SP)
MOV #LNKBLK,-(SP)
EMT 14
```

Il nome globale è: DIR.
Questa richiesta, oltre a restituire i parametri del file richiesto nel directory specificato, indica anche quali funzioni sono possibili sui mezzi senza directories.Il parametro opzionale è usato per indicare se siano richiesti due o tre parametri di ritorno. Questi parametri di ritorno si trovano in testa allo stack nel seguente ordine:

	2 parametri	3 parametri
blocco iniziale		SP
numero di blocchi	SP	SP+2
parola indicativa	SP+2	SP+4

Per numero di blocchi si intende il numero di blocchi del file.
Il formato della parola indicativa è il seguente:

bit	
Ø = 1	è possibile .OPENC
1 = 1	è possibile .OPENI
2 = 1	è possibile .OPENE
3 = 1	è possibile .OPENU
4 = Ø	il file non è in uso
4 = 1	il file è usato da un altro dataset
5 = 1	il dataset ha già un file aperto (nessuna ricerca è stata eseguita)
6 = Ø	il file è concatenato
6 = 1	il file è contiguo
7 = Ø	il file non esiste (è possibile .OPENO)
7 = 1	il file esiste oppure .OPENO non è possibile
8-15	codice di protezione.

L'utente deve togliere dallo stack i parametri di ritorno.
Se un file è protetto contro le operazioni di lettura, sarà segnalato come non esistente.
In caso di errore la richiesta .LOOK si comporta come .ALLOC .

Esempio 1.

Definiamo sul disco nell'unità Ø un file contiguo che occupi 3 blocchi di 256_{10} parole. Il nome del file è PROVA.DAT. Il numero di records introdotti al momento della definizione viene indicato da keyboard in un campo di 5 caratteri. La lunghezza dei records è 16 bytes, ed ogni record è perforato su scheda.

```
        .TITLE ALLOC
        .MCALL .INIT,.ALLOC,.WRITE,.WAIT,.READ,.D2BIN,.OPENC,.CLOSE
        .MCALL .RLSE,.EXIT
PRIMO:
        .INIT #CR
        .INIT #DISK
        .INIT #KBI
        .INIT #KBO
        ;
        ; ALLOCAZIONE DEL FILE
        ;
        .ALLOC #DISK,#FILBLK,#14
        MOV #FILBLK,R0
        ;
        ; SI RICHIEDE DA  VIDEO IL NUMERO DI RECORD
        ;
        .WRITE #KBO,#MES
        .WAIT #KBO
        .READ #KBI,#NRE
        .WAIT #KBI
        .D2BIN #NRE+6
        MOV (SP)+,R1
        TST (SP)+
        TST R1
        BEQ FINE
        .OPENC #DISK,R0
        ;
        ; LETTURA DEL RECORD DA INTRODURRE
        ;
IND:    .READ #CR,#SCHEDA
        .WAIT #CR
        ;
        ; SCRITTURA DEL RECORD
        ;
        .WRITE #DISK,#RECO
        .WAIT #DISK
        SOB R1,IND
        .CLOSE #DISK
```

```
FINE:
        .RLSE #CR
        .RLSE #DISK
        .RLSE #KBI
        .RLSE #KBO
        .EXIT
        .WORD 0
KBI:    .WORD 0
        .RAD50 /KEY/
        .WORD 1
        .RAD50 /KB/
        .WORD 0
KBO:    .WORD 0
        .RAD50 /VID/
        .WORD 1
        .RAD50 /KB/
        .WORD 0
DISK:   .WORD 0
        .RAD50 /FIL/
        .WORD 1
        .RAD50 /DK/
        .WORD 0
        .WORD 0
FILBLK: .RAD50 /PRO/
        .RAD50 /VA/
        .RAD50 /DAT/
        .BYTE 200,200
        .WORD 0
        .WORD 0
CR:   .WORD 0
        .RAD50 /SCH/
        .WORD 1
        .RAD50 /BI/
SCHEDA: .WORD 81.,0,81.
        .BLKB 81.
        .EVEN
MES:    .WORD 100.,0,END-MES-6
        .ASCII <40>/INDICARE IL NUMERO DI RECORDS DA INTRODURRE/<15>
END     =.
        .EVEN
RECO:   .WORD 80.,6,16.
        .WORD SCHEDA+6
NRE:    .WORD 6,0,6
        .BLKB 6
        .EVEN
        .END PRIMO
```

Esempio 2.

Nel programma seguente si fa riferimento al file PROVA.DAT definito nell'esempio precedente.
Si chiede da video all'utente se vuol estendere il file, oppure se vuol leggere un record del file. In tal caso l'utente deve specificare da keyboard il numero del record che vuol leggere. (il primo record ha numero Ø). Il numero del record deve essere scritto in un campo di 5 caratteri e devono essere scritti anche gli zeri non significativi. Il record letto viene riscritto su stampante. Gli eventuali records da aggiungere sono perforati su scheda.

```
 .TITLE ESTENSIONE
        .PSECT PROVA
        .MCALL .INIT,.ALLOC,.OPENC,.READ,.WAIT,.WRITE,.CLOSE
        .MCALL .OPENU,.RECRD,.RLSE,.EXIT
        .MCALL .OPENE
        .MCALL .D2BIN
PRIMO:
        .INIT #CR
        .INIT #DISK
        .INIT #LP
        .INIT #KBI
        .INIT #KBO
        MOV #FILBLK,R0
EST:
        ;
        ; VUOI ESTENDERE IL FILE?
        ;
        .WRITE #KBO,#MES2
        .WAIT #KBO
        .READ #KBI,#ANSW1
        .WAIT #KBI
        CMPB #123,ANSW1+6
        ;
        ;NO - SI PROSEGUE DA CONT
        ;( 123 E' LA CODIFICA ASCII DELLA LETTERA S )
        ;
        BNE CONT
        ;
        ; ALTRIMENTI SI LEGGE IL RECORD E
        ; SI AGGIUNGE AL FILE
        ;
        .READ #CR,#SCHEDA
        .WAIT #CR
        .OPENE #DISK,R0
        .WRITE #DISK,#RECO
        .WAIT #DISK
        .CLOSE #DISK
```

```
CONT:
        ;
        ; VUOI LEGGERE UN RECORD?
        ;
        .WRITE #KBO,#MES3
        .WAIT #KBO
        .READ #KBI,#ANSW2
        ;WAIT #KBI
        CMPB #123,ANSW2+6
        ;
        ; NO - SI PROSEGUE DA END
        ;
        BNE END
        ;
        ; ALTRIMENTI SI RICHIEDE IL NUMERO DI RECORD DA LEGGERE
        ;
        .WRITE #KBO,#MES1
        .WAIT #KBO
        .READ #KBI,#NR
        .WAIT #KBI
        .D2BIN #NR+6
        MOV (SP)+,REC1+10
        TST (SP)+
        .OPENU #DISK,R0
        .RECRD #DISK,#REC1
        .WAIT #DISK
        .WRITE #LP,#REC
        .WAIT #LP
        .CLOSE #DISK
END:
        ; HAI FINITO?
        ;
        .WRITE #KBO,#MES4
        .WAIT #KBO
        .READ #KBI,#ANSW3
        .WAIT #KBI
        CMPB #123,ANSW3+6
        BEQ GIU
        ;
        ; NO - SI TORNA A EST
        ;
        JMP EST
GIU:
        .RLSE #CR
        .RLSE #DISK
        .RLSE #LP
        .RLSE #KBI
        .RLSE #KBO
        .EXIT
        .WORD 0
```

```
KB1:   .WORD 0
       .RAD50 /KEY/
       .WORD 1
       .RAD50 /KB/
       .WORD 0
KBO:   .WORD 0
       .RAD50 /VID/
       .WORD 1
       .RAD50 /KB/
MES1:  .WORD 100.,0,END1-MES1-6
       .ASCII <40>/SCRIVERE IL NUMERO DEL RECORD DA LEGGERE/
       .ASCII <15><12>/:/<15><12>
END1   =.
       .EVEN
NR:    .WORD 6,0,6
       .BLKB 6
       .EVEN
MES2:  .WORD 100.,0,END2-MES2-6
       .ASCII <40>/VUOI ESTENDERE IL FILE ?/
       .ASCII <15><12>/:/<15><12>
END2   =.
       .EVEN
ANSW1: .WORD 3,0,3
       .BLKB 3
       .EVEN
MES3:  .WORD 100.,0,END3-MES3-6
       .ASCII <40>/VUOI LEGGERE UN RECORD ?/
       .ASCII <15><12>/:/<15><12>
END3   =.
ANSW2: .WORD 3,0,3
       .BLKB 3
       .EVEN
MES4:  .WORD 100.,0,END4-MES4-6
       .ASCII <40>/HAI FINITO ?/
       .ASCII <15><12>/:/<15><12>
END4   =.
       .EVEN
ANSW3: .WORD 3,0,3
       .BLKB 3
       .EVEN
       .WORD 0
CR:    .WORD 0
       .RAD50 /SCH/
       .WORD 1
       .RAD50 /B1/
```

```
SCHEDA: .WORD 81.,0,81.
        .BLKB 81.
        .EVEN
        .WORD 0
DISK:   .WORD 0
        .RAD50 /FIL/
        .WORD 1
        .RAD50 /DK/
        .WORD 0
        .WORD 0
FILBLK: .RAD50 /PRO/
        .RAD50 /VA/
        .RAD50 /DAT/
        .BYTE 200,200
        .WORD 0
RECO:   .WORD 80.,6,16.
        .WORD SCHEDA+6
RECI:   .WORD 4
        .WORD BUFF
        .WORD 16.
        .WORD 0
        .WORD 4
        .WORD 0
LP:     .WORD 0
        .RAD50 /RIG/
        .WORD 1
        .RAD50 /LP/
REC:    .WORD 100.,0,<FINE-DUE>
DUE:    .BYTE 12,15
        .ASCII /  CONTENUTO DEL RECORD /
        .BYTE 12
BUFF:   .BLKB 16.
        .BYTE 12,14
FINE:   .BYTE 15
        .EVEN
        .END PRIMO
```

31. LE RICHIESTE PER LA CONVERSIONE DEI DATI

Come abbiamo visto i dati possono essere rappresentati oltre che in binario in codice ASCII e nella forma raggruppata (packed) Radix-5Ø.

Ma se si devono fare dei calcoli l'informazione deve essere binaria, mentre per avere un'uscita sulla stampante i dati devono essere in codice ASCII, decimali o ottali. E' quindi spesso necessario eseguire delle conversioni.

Le macro che eseguono queste conversioni sono:

.RADPK - raggruppa tre caratteri ASCII in una parola in formato Radix-5Ø

.RADUP - converte una parola in formato Radix-5Ø in tre caratteri ASCII

.D2BIN - converte cinque caratteri decimali ASCII in un numero binario di 16 bits

.BIN2D - converte un numero binario di 16 bits in cinque caratteri decimali ASCII

.O2BIN - converte sei caratteri ottali ASCII in un numero binario di 16 bits

.BIN2O - converte un numero binario di 16 bits in sei caratteri ottali ASCII.

Lo schema delle conversioni è il seguente:

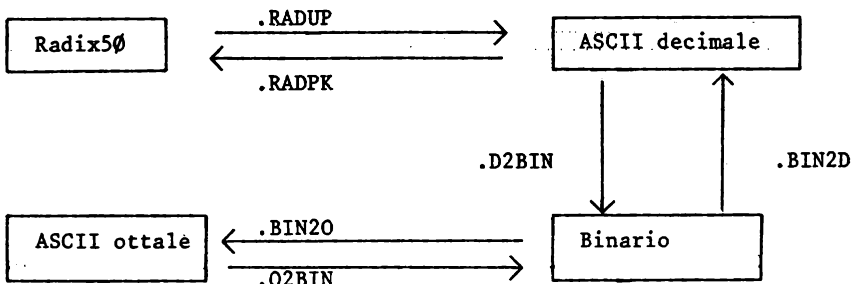


Fig.42-Schema delle conversioni

31.1. .RADPK

La chiamata della macro è:

```
.RADPK #ADDR
```

l'espansione in linguaggio assemblativo è:

```
MOV #ADDR,-(SP)
CLR -(SP)
EMT 42
```

Il nome globale è CVT.
La stringa di 3 caratteri ASCII contenuti in 3 bytes consecutivi a partire dall'indirizzo ADDR è convertita nel formato Radix-5Ø packed. L'algoritmo di conversione è quello visto per la direttiva .RAD5Ø .
Il valore convertito è restituito in testa allo stack seguito dall'indirizzo del byte che segue l'ultimo carattere convertito.
L'utente, quindi, deve liberare lo stack.
Seguendo la formula vista in §17.9., il massimo valore per 3 caratteri è:

$$47 * 5\emptyset^2 + 47 * 5\emptyset + 47 = 174777$$

In caso di errore la conversione viene interrotta e l'utente è informato del tipo di errore attraverso i condition codes nella parola di stato. Se il bit C = 1 significa che è stato incontrato un carattere ASCII non convertibile in Radix-5Ø.
Il valore restituito è rappresentato più a sinistra possibile (left-justified) e corretto fino all'ultimo byte valido.
L'indirizzo restituito sarà quello del byte non accettato
Se non viene incontrato nessun errore durante la conversione, i condition codes saranno azzerati.

Esempio

Vogliamo convertire le ventuno lettere dell'alfabeto in formato Radix-5Ø. I caratteri da convertire si trovano in un'area di memoria che inizia all'indirizzo ASCII; quelli convertiti saranno posti in un buffer all'indirizzo RAD5Ø. In questo esempio ci serviamo del fatto che sullo stack si trova l'indirizzo del byte che segue l'ultimo byte convertito.

```
      MOV #RAD50,R0
      MOV #ASCII,-(SP)          ; .RADPK #ASCII
CONV: CLR-(SP)
      EMT 42
      MOV (SP)+,(R0)+           ; TRASFERISCE IL VALORE CONVERTITO
      CMP R0,#RAD50+16          ; SI CONTROLLA SE TUTTI
                                ; I DATI SONO STATI CONVERTITI
      BNE CONV                  ; NO
      TST (SP)+                 ; SI, ALLORA SI TOGLIE
                                ; IL PUNTATORE DALLO STACK
        .
        .
        .
ASCII: .ASCII /ABCDEFGHILMNOPQRSTUVZ/
RAD50: .BLKW 7
        .
        .
```

31.2. .RADUP

La chiamata della macro è:

.RADUP #ADDR,WORD

dove WORD è la parola Radix-5Ø da convertire e ADDR è l'indirizzo del primo dei 3 bytesche conterranno i caratteri ASCII.
L'espansione in linguaggio assemblativo è:

```
MOV WORD,-(SP)
MOV #ADDR,-(SP)
MOV #1,-(SP)    ; MOVE CALL CODE ONTO STACK
EMT 42
```

Il nome globale è CVT.
Il contenuto di WORD è convertito in una stringa di caratteri ASCII, a 7-bits,che sono posti, left-justified, in 3 bytes consecutivi a partire dall'indirizzo ADDR.
Eventuali errori vengono segnalati ponendo a 1 il bit C della PS. sono possibili due tipi di errore:

a) il valore della parola da convertire è > 174777

b) il valore di un byte Radix-5Ø risulta 35, che correntemente non è usato.

In caso di errore di tipo a) il primo dei 3 bytes conterrà il carattere ":",mentre per un errore di tipo b) ciascuno dei 3 bytes conterrà una / .
Se non si verificano errori, i condition codes sono azzerati.

Esempio.

```
        .RADUP #ABC,RAD50
        .
        .
        .
ABC:    .BLKB 3
        .
        .
```

converte da formato Radix-5Ø in ASCII il contenuto di RAD5Ø.
I caratteri ASCII vengono messi nei 3 bytes all'indirizzo iniziale ABC.

31.3. .D2BIN

La macro .D2BIN richiede come parametro l'indirizzo del buffer di 5 bytes che contiene i caratteri decimali ASCII da convertire, per cui la chiamata della macro è:

```
.D2BIN #ADDR
```

L'espansione in linguaggio assemblativo è:

```
MOV #ADDR,-(SP)
MOV #2,-(SP)    ; MOVE CALL CODE ONTO STACK
EMT 42
```

Il nome globale è CVT.
La stringa di 5 caratteri decimali ASCII che inizia all'indirizzo ADDR è convertita nell'equivalente binario. Il valore convertito si trova in testa allo stack seguito dall'indirizzo del byte che segue l'ultimo carattere convertito. Il più grande numero decimale che può essere convertito è 65535 (2^{16}-1). In caso di errore la conversione è interrotta.
Se il bit C è 1 significa che si è incontrata una situazione di errore causata dal fatto che un byte non è una cifra decimale.
Se il bit V è 1 significa che il numero decimale era troppo grande.
Se durante la conversione non si incontrano errori i condition codes sono azzerati.

Esempio

Per convertire 4 numeri costituiti ciascuno da 5 cifre decimali e contenuti nel buffer all'indirizzo DEC si può usare il seguente segmento di programma. I dati binari vengono trasferiti nel buffer all'indirizzo BIN. Anche in questo esempio di tiene conto del fatto che sullo stack si trova l'indirizzo del byte che segue l'ulti-

mo byte convertito.

```
        MOV #BIN,R1
        MOV #DEC,-(SP)          ; .D2BIN #DEC
CON:    MOV #2,-(SP)
        EMT 42
        MOV (SP)+,(R1)+         ; TRASFERISCE NEL BUFFER
                                ; I DATI BINARI
        CMP R1,#BIN+10          ; SI SONO CONVERTITI TUTTI I DATI ?
        BNE CON                 ; NO
        TST (SP)+               ; SI, ALLORA SI TOGLIE IL
                                ; PUNTATORE DALLO STACK
          .
          .
          .
DEC:    .BLKB 20.
BIN:    .BLKW 4
          .
          .
```

31.4. .BIN2D

La macro .BIN2D richiede due parametri: l'indirizzo del primo byte che conterrà i dati decimali ASCII e la parola che contiene il numero binario da convertire.
La chiamata è pertanto:

```
.BIN2D #ADDR,WORD
```

e l'espansione in linguaggio assemblativo è:

```
MOV WORD,-(SP)
MOV #ADDR,-(SP)
MOV #3,-(SP)    ; MOVE CALL CODE ONTO STACK
EMT 42
```

Il nome globale è CVT.
Il contenuto di WORD è convertito in 5 decimali ASCII che sono posti in bytes consecutivi a partire dall'indirizzo ADDR.
I caratteri ASCII sono appoggiati a destra (righ-justified),con in testa degli zero.

Osservazione

Come abbiamo visto(cfr.§22.)i parametri delle macro possono essere passati usando quasi tutti i modi di indirizzamento.
Per eseguire la conversione il Monitor si aspetta di avere sullo stack il valore da convertire e l'indirizzo del buffer. Pertanto il secondo parametro della macro .BIN2D può essere il numero stesso da convertire, invece del nome simbolico della posizione che lo contiene.
Per esempio se il numero binario 10000 è contenuto in WORD, la chiamata della macro .BIN2D nella forma:

```
.BIN2D #ADDR,#↑B10000
```

è equivalente a:

```
.BIN2D #ADDR,WORD
```

La stessa osservazione vale per le macro .RADUP e .BIN2O.

Esempio

Nel seguente esempio sono illustrati due modi possibili della chiamata della macro..BIN2D. Nel primo il contenuto di B2 viene convertito in una stringa di 5 decimali ASCII posti in 5 bytes consecutivi a partire dall'indirizzo SOM; nel secondo viene convertito in decimale ASCII il numero binario 10101

```
        .BIN2D #SOM,B2
        .BIN2D #SOM1,#^B10101
          .
          .
          .
SOM:    .BLKB 5
B2:     .BLKW 1

SOM1:   .BLKB 5
```

31.5. .O2BIN

Questa macro richiede come unico parametro l'indirizzo del primo dei 6 bytes che contengono i caratteri ottali ASCII:

.O2BIN #ADDR.

L'espansione in linguaggio assemblativo è:

```
MOV #ADDR,-(SP)
MOV #4,-(SP) ;MOVE CALL CODE ONTO STACK.
EMT 42
```

Il nome globale è CVT.
La stringa di sei caratteri ASCII ottali che inizia all'indirizzo ADDR è convertita nel numero binario equivalente.
Il valore convertito si trova in testa allo stack seguito dall'indirizzo del byte che segue l'ultimo byte convertito.
Se si verifica un errore la conversione è interrotta e il tipo di errore è segnalato attraverso i bits condition codes nella PS:

C = 1 significa che un byte non era una cifra ottale.

V = 1 significa che il numero da convertire era troppo grande, cioè il primo byte era maggiore di 1. Il più grande numero ottale che può essere convertito è, infatti 177777.

31.6. .BIN2O

La macro .BIN2O richiede come parametri l'indirizzo del primo dei 6 bytes che conterrano i caratteri ottali e la parola binaria da convertire:

.BIN2O #ADDR,WORD

L'espansione in linguaggio assemblativo è:

```
MOV WORD,-(SP)
MOV #ADDR,-(SP)
MOV #5,-(SP)
EMT 42
```

Il contenuto binario di WORD è convertito in una stringa di 6 caratteri ottali ASCII contenuti in 6 bytes consecutivi a partire dall'indirizzo ADDR.
(cfr. Osservazione in 31.4)

Esempio 1.

Si voglia la rappresentazione ottale di un insieme di dati decimali. I dati sono perforati uno per scheda nel primo campo di 5 colonne. Poiché non esistono macro che permettano la conversione direttamente da decimale ASCII a ottale ASCII, i dati devono prima essere convertiti in binario e poi da binario a ottale. Il pacco di dati termina con una scheda che contiene il carattere @ nella prima colonna.

```
        .TITLE CONVE
        .SBTTL CONVERSIONE DA DECIMALE A OTTALE
        .MCALL .INIT,.READ,.WAIT,.WRITE,.BIN2O,.D2BIN,.RLSE,.EXIT
PRIMO:
        .INIT #CR
        .INIT #LP
        .WRITE #LP,#MES
        .WAIT #LP
LEGGI:
        .READ #CR,#DEC              ; INGRESSO DATI
        .WAIT #CR
        CMPB #100,DEC+6             ; HA LETTO TUTTI I DATI ?
        BEQ FINE                    ; SI - VAI A FINE
        MOV #DEC+6,R0
        MOV #DECI,R1
TRASF:
        MOVB (R0)+,(R1)+            ; TRASFERISCE NEL BUFFER
                                    ; DI USCITA IL DATO DECIMALE LETTO
        CMP R0,#DEC+13
        BNE TRASF
        .D2BIN #DEC+6               ; CONVERTE DA DECIMALE ASCII A BINARIO
        MOV (SP)+,BIN
        TST (SP)+
        .BIN2O #OTT,BIN             ; CONVERTE DA BINARIO A
                                    ; OTTALE ASCII
        .WRITE #LP,#CONV            ; STAMPA
        .WAIT #LP
        BR LEGGI
FINE :
        .RLSE #CR
        .RLSE #LP
        .EXIT
```

```
       ; DEFINIZIONE DEI FILE DI I/O
       .WORD 0
CR:    .WORD 0
       .RAD50 /INP/
       .WORD 1
       .RAD50 /BI/
       .WORD 0
LP:    .WORD 0
       .RAD50 /OUT/
       .WORD 1
       .RAD50 /LP/
DEC:   .WORD 81.,0,81.
       .BLKB 81.
       .EVEN
CONV:  .WORD 100.,0,END -CONV-6
       .BYTE 40,40,40,40,40
DECI:  .BLKB 5
       .BYTE 40,40,40,40,40,40,40,40,40,40,40,40,40,40,40
OTT:   .BLKB 6
       .BYTE 12,12,12,12,12
END=.
       .EVEN
BIN:   .BLKW 1
MES:   .WORD 100.,0,MESS-MES-6
       .BYTE 12,12
       .ASCII <40><40><40>/VALORE DECIMALE/
       .ASCII <40><40><40><40>/VALORE OTTALE/
    .BYTE 12,12,12
MESS   =.
       .EVEN
       .END PRIMO
```

Esempio 2.

Consideriamo il problema seguente: un pacco di schede contiene dei dati sperimentali rappresentati in quattro campi decimali di 5 colonne e un campo ottale di 6 colonne.
La fine dei dati è segnalata da una scheda che contiene nel primo campo una @ .
Vogliamo che sia stampata, per ogni scheda, la media dei quattro campi decimali e la somma totale dei campi ottali. I dati decimali dovranno essere, quindi, convertiti in binario per farne la media e poi di nuovo in decimale ASCII per la stampa. La stessa procedura dovrà essere seguita per i campi ottali, (le somme parziali sono contenute nella locazione ausiliaria OTT).

```
        .TITLE ESEMPIO
        .PSECT ES
        .MCALL .READ,.WRITE,.INIT,.WAIT,.RLSE,.EXIT,.D2BIN,.BIN2D
        .MCALL O2BIN,.BIN2O
PRIMO:  .INIT #LEGGI              ; SI INIZIALIZZANO I MEZZI DI I/O
        .INIT #SCRIVI
        CLR R3
CONT:
        INC R3                    ; CONTATORE DI SCHEDE
        .READ #LEGGI,#NUM         ; LETTURA
        .WAIT #LEGGI
        CMPB #100,DECI            ; IL PRIMO CARATTERE E' UNA @ ?
        BEQ TOT                   ;NO VAI A TOT ALTRIMENTI IN SEQUENZA
        .BIN2D #SCHE,R3           ; CONVERSIONE A DECIMALE DEL
                                  ; NUMERO DI SCHEDE
        JSR R5,ZERO               ; SI RICHIAMA IL SOTTOPROGRAMMA
        .WORD SCHE                ; ZERO PER TOGLIERE GLI ZERI NON
                                  ; SIGNIFICATIVI NEL NUMERO DI SCHEDE
        MOV #BIN,R1               ; TRASFERISCE IN R1 L'INDIRIZZO DEL
                                  ; BUFFER BIN CHE CONTERRA' I DATI BINARI
        MOV #DECI,-(SP)           ; .D2BIN #DECI
CONV:   MOV #2,-(SP)
        EMT 42
        MOV (SP)+,(R1)+           ; TRASFERISCE NEL BUFFER I DATI BINARI
        CMP R1,#BIN+10            ; SONO STATI CONVERTITI TUTTI I DATI ?
        BNE CONV                  ; NO
        MOV #4,-(SP)              ; SI-CONVERTE DA OTTALE ASCII A BINARIO
        EMT 42                    ; (.O2BIN #DECI+24) TENENDO PRESENTE
```

```
                                ; CHE IN TESTA ALLO STACK SI TROVA
                                ; L' INDIRIZZO DEL PRIMO DEI SEI BYTES
                                ; CHE CONTENGONO I CARATTERI OTTALI
        MOV (SP)+,R0            ; TRASFERISCE  L' EQUIVALENTE BINARIO
                                ; DEL CAMPO OTTALE IN R0
        ADD R0,OTT              ; ESEGUE LA SOMMA DEI CAMPI OTTALI
        ADD BIN+2,BIN           ; SI ESEGUE LA SOMMA DEI
        ADD BIN+4,BIN           ; PRIMI QUATTRO CAMPI
        ADD BIN+6,BIN
        ASR BIN                 ; SI CALCOLA LA MEDIA DIVIDENDO
        ASR BIN                 ; PER QUATTRO IL CONTENUTO DI BIN
        .BIN2D #SOM,BIN         ; SI CONVERTE IN DECIMALE IL CONTENUTO
                                ; DI BIN E SI PONE NEL BUFFER SOM
        JSR R5,ZERO             ; SI TOLGONO GLI ZERI NON SIGNIFICATIVI
        .WORD SOM               ; NELLA SOMMA
        .WRITE #SCRIVI,#ASS     ; STAMPA DEL NUMERO DI ORDINE
                                ; DELLA SCHEDA E DELLA MEDIA
        .WAIT #SCRIVI
        BR CONT
TOT:
        .BIN2O #OTTA,OTT        ; CONVERTE DA BINARIO A OTTALE
        .WRITE #SCRIVI,#OT      ; STAMPA  IL TOTALE DEI CAMPI OTTALI
        .WAIT #SCRIVI
        .RLSE #LEGGI
        .RLSE #SCRIVI
        .EXIT
;
; SOTTOPROGRAMMA ZERO
;
ZERO:   MOV R1,-(SP)            ; SALVA IL CONTENUTO DEI REGISTRI
        MOV R2,-(SP)
        CLR R2                  ; AZZERA R2 CHE SERVE DA CONTATORE
        MOV (R5)+,R1            ; R1 CONTIENE L' INDIRIZZO DEL BUFFER
                                ; DEI DATI ASCII
BLK:    INC R2
        CMPB #060,(R1)          ; IL CARATTERE IN ESAME E' 0 ?
        BNE FINE                ; NO - VAI  A FINE
        CMP #5,R2               ; SI - HAI RAGGIUNTO L' ULTIMO
                                ; CARATTERE ?
        BEQ FINE                ; SI - VAI A FINE
        MOVB #40,(R1)+          ; NO - SOSTITUISCILO
                                ; CON UN CARATTERE VUOTO
        BR BLK
FINE:   MOV (SP)+,R2            ; RIPRISTINA IL CONTENUTO DEI REGISTRI
        MOV (SP)+,R1
        RTS R5
```

```
;
; DEFINIZIONE DEI FILES DI I/O
;
        0
LEGGI : 0
        .RAD50 /INP/
        1
        .RAD50 /BI/
        0
SCRIVI: 0
        .RAD50 /OUT/
        1
        .RAD50 /LP/
NUM: 81.,0,81.
DECI:   .BLKB 81.
        .EVEN
BIN:    .BLKW 4
ASS: 130.,0,<STAMP-STAM>
STAM: .BYTE 14,15
        .ASCII /    SCHEDA NUMERO  /
SCHE: .BLKB 5
        .BYTE 15,12,12
        .ASCII /     LA MEDIA E'    /
SOM:    .BLKB 5
        .BYTE 15,12,12
STAMP: .BYTE 15
        .EVEN
OT:     .WORD 100.,0,<UNO-DUE>
DUE:    .BYTE 15,12,12
        .ASCII /    LA SOMMA TOTALE DEI CAMPI OTTALI E'   /
        .BYTE 12
OTTA: .BLKB 6
        .BYTE 14
UNO:    .BYTE 12
        .EVEN
OTT: .WORD 0
        .END PRIMO
```

32. LA RICHIESTA .EXIT

La richiesta .EXIT restituisce il controllo al Monitor. Questa macro non richiede alcun parametro, quindi viene richiamata semplicemente con la frase:

.EXIT

L'espansione in linguaggio asemblativo è:

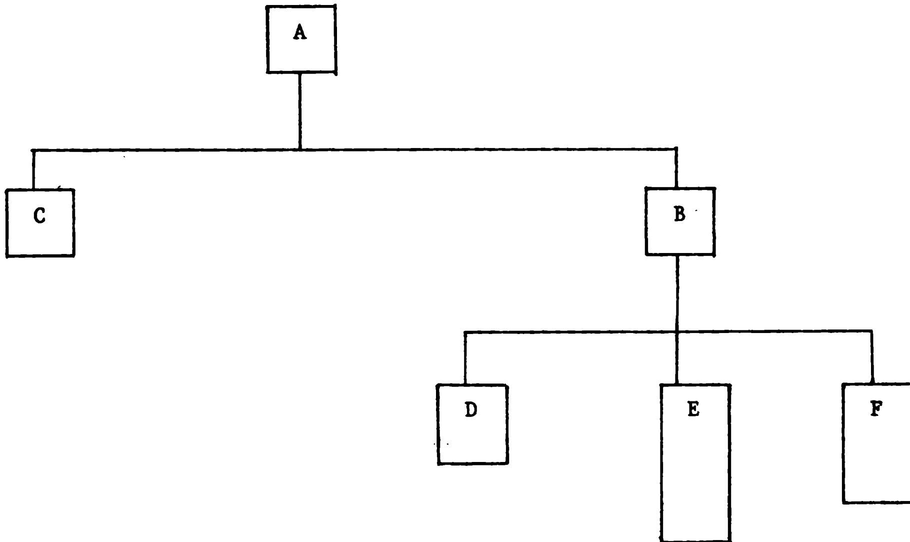
EMT 6Ø

e il nome globale è: XIT.
Questa è l'ultima frase eseguita in un programma utente, ossia indica la fine dell'esecuzione di un programma. Il controllo è restituito al Monitor, assicurando che tutti i files di dati del programma siano stati chiusi e in generale preparando per accettare il successivo comando.
Tutti gli spazi riservati per il programma sono liberati.

33. LA RICHIESTA .RUN

Un programma molto lungo può essere diviso in una serie di segmenti più corti. I segmenti possono essere caricati in memoria tutti insieme ma se il programma è troppo lungo per essere tutto contenuto in memoria i segmenti possono essere letti in sequenza. Nel secondo caso i segmenti o gruppi di segmenti caricati insieme sono detti "overlays".
Con la tecnica di overlay la stessa area di memoria è usata per contenere in tempi successivi parti differenti del programma. Ciascuna parte del programma che deve spartire la stessa area di overlay è caricata nell'area per mezzo di un programma di controllo.

Gli overlays sono descritti con una semplice struttura ad albero in cui la radice è detta *segmento radice* e rimane sempre in memoria; i rami dell'albero rappresentano i *segmenti overlays*. Per esempio una tipica struttura di overlay è:



dove A è il segmento radice, e B,C,D,E,F sono i segmenti overlay. Per caricare un overlay, o un intero programma, esiste la macro .RUN .
La macro .RUN ha come unico parametro l'indirizzo di una tabella, detta Run Block, che contiene le informazioni necessarie:

```
.RUN #RUNBLK
```

L'espansione in linguaggio assemblativo è:

```
MOV #RUNBLK,-(SP)
EMT 65
```

Il nome globale è RUN.
Si possono, con questa macro, specificare parecchie opzioni fra le quali:

- caricare un programma o un overlay;
- caricare una copia di memoria o un modulo eseguibile;
- stabilire dove deve ritornare il controllo, cioè l, istruzione successiva da eseguire;
- muovere, o meno, lo stack;
- indicare l'indirizzo di caricamento.

Tutte queste opzioni sono specificate nella prima parola del Run Block. Questo è una tabelladi lunghezza variabile; l'unica parola sempre presente è quella che contiene l' informazione sulla funzione da eseguire, mentre le altre parole possono essere omesse, sotto certe condizioni.
Una di queste parole contiene l'indirizzo del Link Block che descrive il mezzo da cui l'entità deve essere caricata.
La presenza del Link Block è necessaria a meno che il bit 15 della parola del Run Block che indica la funzione da eseguire sia 1; tuttavia il Link Block non deve essere inizializzato.
Se è presente il Link Block, dovrà essere stabilito anche un File Block, interno al programma, il cui indirizzo sarà contenuto in un'altra parola del Run Block. Il file non deve tuttavia essere aperto.
La richiesta .RUN toglie l'indirizzo del Run Block dallo stack; se però il bit Ø della voce che indica la funzione è Ø in testa allo stack si troveranno le seguenti informazioni:

(SP) -indirizzo di trasferimento del modulo eseguibile
2(SP) -misura del modulo in bytes
4(SP) -indirizzo più basso del modulo eseguibile.

Per maggiori dettagli cfr. manuale [2].

PARTE QUARTA
CENNI SUL SISTEMA DI "INTERRUPT"

Questa parte contiene alcune informazioni di carattere generale sul sistema di interruzioni. Per maggiori informazioni cfr.[1].

34. INTRODUZIONE

L'unità di processo, la memoria e tutti i periferici sono connessi con un solo, comune insieme di linee, detto UNIBUS. Le comunicazioni fra due mezzi sul canale (bus) sono regolate da una relazione di *padrone-servo*. Durante ogni operazione sul bus, infatti, un mezzo, chiamato appunto il *padrone* del bus controlla il bus e comunica con un altro mezzo sul bus, chiamato il *servo* . Un tipico esempio di questa relazione è l'unità di processo, come padrone, che trasferisce dati alla memoria, come servo. Questa relazione è, tuttavia, dinamica.
L'unità di processo, infatti, può passare il controllo per esempio, ad un disco, che quindi diventa il nuovo padrone.
L'UNIBUS è usato dall'unità di processo e da tutti i mezzi di ingresso/uscita, ossia l'unità di processo e i periferici possono ottenere il controllo del bus. Poiché più di un mezzo può richiedere il controllo del bus nello stesso momento, è necessario stabilire una priorità. Di conseguenza ad ogni mezzo che possa ottenere il controllo del bus è assegnato un livello di priorità. Quando due mezzi con lo stesso livello di priorità richiedono contemporaneamente l'uso del bus, il controllo passa al mezzo che è più vicino all'unità di processo.
Un mezzo (oltre all'unità di processo) capace di diventare il padrone del bus, generalmente richiede l'uso del bus per uno dei due scopi:

a) per eseguire un trasferimento di dati direttamente, senza il controllo dell'unità di processo, in o dalla memoria;
b) interrompere l'esecuzione del programma e forzare l'unità di processo e passare il controllo ad un indirizzo specifi-

cato, che costituisce il punto di ingresso di una *routine di servizio* .

35. L'ORGANIZZAZIONE E IL CONTROLLO DEI MEZZI PERIFERICI

Le funzioni di controllo dei periferici sono assegnate a registri (indirizzabili come le altre locazioni di memoria) ossia le operazioni di controllo sono determinate dal valore dei singoli bits all'interno di questi registri.
Per esempio il comando per far leggere una scheda dal lettore è espresso ponendo a 1 un bit (il bit Ø) nel registro di controllo del mezzo.
Analogamente, lo stato del mezzo è manipolato con assegnazioni di bits all'interno del registro, e può essere controllato da istruzioni del programma.
Vi sono due tipi di registri associati con ciascun mezzo:

a) i registri di stato e controllo

b) i registri che contengono dati.

I registri di controllo e stato contengono tutte le informazioni necessarie per comunicare con quel mezzo. Il formato reale di tali registri dipende dal mezzo, ma può essere schematizzato nel modo seguente:

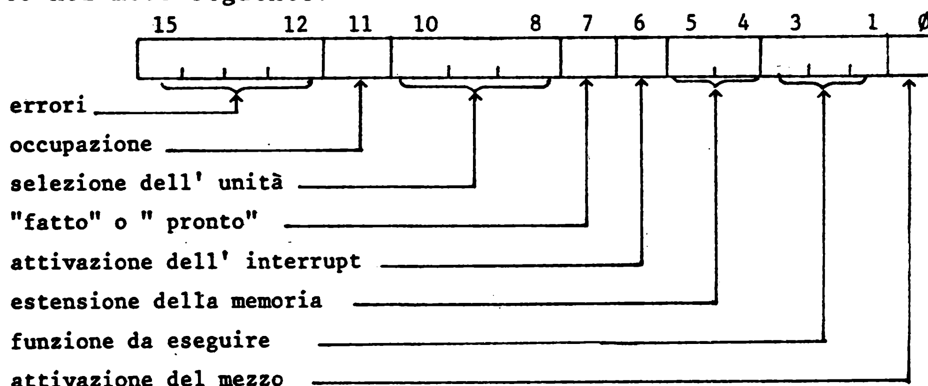


Fig.43-Formato generale di un registro di stato

bits		funzione
15-12	errori	generalmente ciascun bit è associato con uno specifico errore, e il bit 15 è l'OR inclusivo di tutti gli altri bits di errore;
11	occupazione	questo bit è = 1 quando il mezzo sta eseguendo un' operazione;
10-8	selezione dell'Unità	questi bits servono per specificare l'unità richiesta, ad esempio la superficie di un disco ;
7	"fatto" o "pronto"	questi bits sono azzerati o messi a 1 dal periferico ma possono essere controllati a programma per stabilire la disponibilità del mezzo;
6	attivazione interrupt	questo bit è messo a 1 a programma per permettere che avvenga un' interruzione dopo l'esecuzione di una funzione o in caso di errore;
5-4	estenzione della memoria	permettono ai periferici di usare 18 bits per specificare gli indirizzi sul bus.
3-1	funzione	specificano l'operazione che un mezzo deve eseguire;
∅	attivazione	viene messo a 1 per attivare un periferico ;

Alcuni mezzi possono richiedere più di 16 bits e perciò saranno necessari più registri di controllo.
Alcuni bits sono controllati soltanto dal periferico, cioè il programma può controllare questi bits ma non può modificarli.
Altri bits possono essere, invece, solo scritti dal programma e sono sempre letti come zero.

I registri di tipo b) sono usati per memorizzare temporaneamente i dati da trasferire. Anche il numero e il tipo di questi registri dipende dal mezzo.
Per esempio i dati da far uscire sulla stampante devono essere trasferiti un carattere alla volta nel registro all'indirizzo 777516, in modo che il byte di destra di tale registro contenga il codice ASCII del carattere da stampare.

Ciascun mezzo periferico ha inoltre, un puntatore hardware ad una coppia di posizioni di memoria (una coppia per ciascun mezzo) che forma il così detto *vettore di interrupt* .
La prima di queste due parole conterrà l'indirizzo della *routine di servizio* del mezzo(*); l'altra conterrà le informazioni sul nuovo stato dell'unità di processo, cioè le informazioni relative all'*ambiente* in cui agirà la routine di servizio.
La priorità di interrupt dei periferici è indipendente dalla priorità della routine di servizio.
La priorità della routine di servizio può essere cambiata a programma, facendo sì quindi che il comportamento del sistema dipenda da certe condizioni.
Il sistema di interrupt permette all'unità di processo di confrontare continuamente la sua priorità programmabile con la priorità di ogni mezzo capace di provocare un' interruzione e soddisfare la richiesta del mezzo con priorità più alta della propria.

(*) dove la *routine di servizio* è un segmento di programma scritto dall'utente per manipolare i dati e i registri di stato

36. LA STRUTTURA DI PRIORITA'

Come abbiamo visto, l'uso del bus è concesso ai mezzi richiedenti secondo un certo schema di priorità.
La priorità di un mezzo è una funzione del livello di priorità assegnato al mezzo e della sua posizione sul bus rispetto ad altri mezzi con lo stesso livello di priorità.
Ad ogni mezzo, ad eccezione, dell'unità di processo può essere assegnato uno dei cinque livelli di priorità; a ciascuno di questi livelli è collegata una linea di segnali. Queste cinque linee sono chiamate *linee di richiesta* , e sono controllate da un *arbitro* .
Un mezzo che richiede l'uso del bus fa una richiesta (BR) su una di queste linee. Questa richiesta è ricevuta dall'arbitro che controlla anche la priorità dell'unità di controllo.
Se non è pervenuta alcuna richiesta ad un livello di priorità più alto del livello corrente dell'unità di processo, l'uso del bus rimane all'unità di processo. Altrimenti l'arbitro emette un segnale di concessione (BG) al livello più alto delle richieste attive. Il segnale di concessione è ricevuto dal primo mezzo sul bus assegnato allo stesso livello di priorità del segnale. Se questo mezzo è quello che ha richiesto l'uso del bus blocca il segnale di concessione ed informa di averlo ricevuto. Altrimenti passa il segnale al mezzo successivo assegnato alla stessa linea di livello. Questo procedimento è ripetuto finché un mezzo accetta il segnale.
Consideriamo ad esempio lo schema illustrato nella figura seguente:

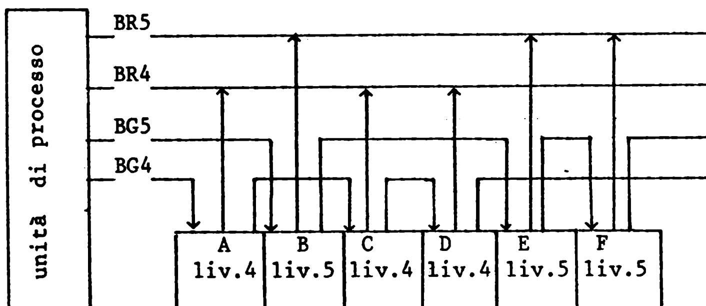


Fig.44-Schema di priorità

Tre mezzi sono a livello di priorità 4: A,C,D; gli altri tre mezzi, B,E,F, sono a livello di priorità 5.
Se il livello di priorità dell'unità di processo è maggiore o uguale a 5, non viene soddisfatta nessuna richiesta dell'uso del bus fatta da questi periferici.
Supponiamo che il livello di priorità dell'unità di processo sia 2, ed inoltre che durante l'esecuzione di una istruzione i mezzi C,E e F richiedano l'uso del bus. Alla fine dell'istruzione l'unità di processo controlla le richieste. Poiché c'è una richiesta a livello 5, l'unità di processo non risponde al periferico C. Il segnale di risposta al livello 5 arriva prima al mezzo B, ma prosegue oltre poiché il mezzo B non ha fatto alcuna richiesta, e quindi non blocca l'impulso.
Il segnale arriva poi al mezzo E che lo blocca e prende il controllo del bus.
Tuttavia i mezzi F e C hanno ancora attive le loro richieste.
Se il mezzo E segnala che deve essere eseguito un interrupt, il periferico F ottiene il controllo del bus dopo che la prima istruzione della routine di servizio è stata eseguita, a meno che la priorità dell'unità di processo non sia stata alzata.
Cambiare la priorità dell'unità di processo è abbastanza semplice in quanto dopo che un mezzo ha segnalato la richiesta di esecuzione di un interrupt viene caricata una nuova parola di stato che contiene la nuova priorità dell'unità di controllo.
Se tale priorità è stata portata a livello 5 viene ignorata la richiesta del mezzo F.
Alla conclusione della routine di servizio viene ristabilita la priorità originale dell'unità di controllo e ripreso il procedimento normale. Dopo una istruzione il periferico F ottiene il controllo del bus. Infine quando il procedimento normale viene di nuovo ripreso il periferico C, che sta ancora aspettando, ottiene il controllo del bus in modo analogo.

Da quanto detto, risulta che ad ogni mezzo sull'UNIBUS è asse-

gnata una posizione discreta nello schema di priorità. Questa posizione è determinata:

a) dal livello di priorità assegnato al mezzo e
b) dalla posizione del mezzo sulla linea di risposta, rispetto agli altri mezzi allo stesso livello di priorità.

Quindi all'interno di un dato livello di priorirà il mezzo più vicino all'origine del segnale di risposta ha la più alta priorità effettiva. Le priorità più alte sono assegnate a mezzi che richiedono un servizio più veloce per evitare la distruzione o la perdita di dati.

37. IL TRASFERIMENTO DEI DATI

La richiesta del bus da parte di mezzi esterni può essere fatta per eseguire un trasferimento di dati.
Il trasferimento di dati può avvenire fra due periferici senza il controllo dell'unità di processo. Questi sono chiamati trasferimenti a livello NPR e normalmente sono fatti fra la memoria e i mezzi con memoria di massa, come ad esempio un disco.
La priorità più alta è assegnata alle richieste a livello NPR.
Un mezzo in cui siano possibili trasferimenti a livello NPR ha un accesso molto veloce al bus. Lo stato dell'unità di processo non è interessato dal trasferimento e perciò questa può rilasciare il controllo mentre sta eseguendo un'istruzione. Questo passaggio di controllo del bus può avvenire, in generale, ogni qual volta l'unità di controllo non stia usando il bus.

38. LE RICHIESTE DI INTERRUPT

I mezzi che ottengono il controllo del bus con una delle linee di richiesta (BR7,BR6,BR5,BR4) possono sfruttare tutti i vantaggi della potenza e flessibilità dell'untià centrale di

controllo richiedendo un'*interrupt*.
Quando deve essere mandato in esecuzione un programma di servizio per un mezzo, il lavoro che l'unità centrale stava eseguendo è interrotto ed è iniziata la routine di servizio del periferico. Dopo che la richiesta del mezzo è stata soddisfatta l'unità di controllo riprende il lavoro precedentemente interrotto.
E' bene tener presente che le richieste di interrupt possono essere fatte soltanto se il controllo del bus è stato ottenuto attraverso un livello di priorità BR, cioè una richiesta a livello NPR non può essere usata per una richiesta di interrupt.
Le operazioni richieste per servire un mezzo sono le seguenti:

a) priorità permettendo, l'unità di processo lascia il controllo del bus al mezzo;

b) quando il mezzo ha ottenuto il controllo del bus, invia all'unità di processo un comando di interrupt e un determinato indirizzo di una posizione di memoria che contiene il punto di ingresso della routine di servizio, cioè l'indirizzo del vettore di interrupt. La parola seguente che si trova cioè all'indirizzo del vettore +2, è usata come la nuova parola di stato, (PS).

c) l'unità di processo pone in testa allo stack la parola di stato corrente e poi il valore del PC.

I nuovi PC e PS sono presi dall'indirizzo specificato dal mezzo (vettore di interrupt) e la routine di servizio è iniziata.
La sequenza di istruzioni eseguita, a livello hardware, al verificarsi di un interrupt, può quindi essere esemplificata nel modo seguente:

```
MOV OLDPS,-(PS)
MOV OLDPC,-(SP)

MOV SVRT,PC
MOV NEWPSW,PS
```

dove SVRT è il punto di ingresso della routine di servizio per il mezzo e che si trova nella prima voce del vettore di interrupt e NEWPSW è la nuova parola di stato.
Queste operazioni sono eseguite automaticamente e non è necessario selezionare i periferici per stabilire quale routine di servizio si debba eseguire.

e) Se la routine di servizio esegue una istruzione di ritorno dall'interrupt (RTI), l'unità di controllo riprende il processo interrotto togliendo le due parole in testa allo stack e trasferendole nei registri PC e PS.

Una routine di servizio può a sua volta essere interrotta da una richiesta con priorità più alta, in ogni momento dopo che la prima istruzione della routine è stata eseguita. In tal caso il PC e PS della routine corrente sono posti sullo stack e viene iniziata, con lo stesso procedimento, la nuova routine di servizio.

39. LE ISTRUZIONI PER GLI INTERRUPTS

Gli interrupts sono generalmente trasparenti per il programmatore, ma esistono alcune istruzioni che permettono all'utente di gestire a programma gli interrupts.
Tali istruzioni sono illustrate in questo paragrafo.

► EMT	emulator trap
codice ottale	1ØØ4ØØØ-1Ø4377

Questa istruzione causa una interruzione gestita tramite la posizione all'indirizzo 3Ø, ossia il nuovo valore del PC è preso dalla parola all'indirizzo 3Ø e il nuovo stato (PS) è preso dalla parola all'indirizzo 32.
Tutte le istruzioni con codice da 1Ø4ØØØ e 1Ø4377 sono istruzioni EMT e sono usate per trasmettere informazioni alla funzione da

eseguire(cfr.ad esempio le macro di sistema).
Poiché l'istruzione EMT è frequentemente usata dal software di sistema, non è opportuno usarla. Il valore dei bit N,Z,V,C è determinato dalla parola all'indirizzo 32.

► TRAP trap

codice ottale 1Ø44ØØ-1Ø4777

Le istruzioni con codice da 1Ø44ØØ a 1Ø4777 sono istruzioni TRAP. Queste istruzioni sono identiche dal punto di vista operativo alle istruzioni EMT eccetto che il vettore di interrupt si trova all'indirizzo 34.
L'istruzione TRAP deve essere preceduta da una chiamata della macro di sistema .TRAP..Tale macro di sistema ha due parametri:

```
.TRAP #STATUS,#ADDR
```

dove STATUS è l'indirizzo del nuovo stato dell'unità di controllo e ADDR è il punto di ingresso della routine di servizio.
L'espansione in linguaggio assemblativo è:

```
MOV #ADDR,-(SP)
MOV #STATUS,-(SP)
MOV #1,-(SP)       ; 1 è il codice di identifi-
                     cazione per .TRAP
EMT 41
```

il nome globale è GUT.
Questa macro mette STATUS e ADDR nel vettore di interrupt all'indirizzo 34. Dopo che l'istruzione è completata il controllo ritorna all'utente all'istruzione che segue l'espansione in linguaggio assemblativo e i parametri sono tolti dallo stack.
L'utente può poi usare l'istruzione TRAP, che trova quindi agli

indirizzi 34 e 36 le informazioni necessarie per eseguire la interruzione.

► BPT breakpoint trap

codice ottale ØØØØØ3

Questa istruzione esegue una sequenza di interrupt con vettore di interrupt all'indirizzo 14. Questa istruzione è usata per richiamare i segmenti di programma usati per controllare i programmi stessi, cioè le routines che servono per la messa a punto dei programmi.

► IOT input/output trap

codice ottale ØØØØØ4

Esegue una sequenza di interrupt con vettore di interrupt all'indirizzo 2Ø. Nei sistemi DOS questa istruzione è usata per segnalare gli errori.

► RTI return from interrupt

codice ottale ØØØØØ2

Questa istruzione è usata per uscire da una routine di servizio. I registri PC e PS sono ripristinati togliendoli dallo stack. E' opportuno ricordare che se il bit 4 (bit T) nella parola di stato dell'unità di controllo è 1, avviene un'interruzione con vettore di interrupt all'indirizzo 14 quando l'esecuzione dell'istruzione è completata. Questo bit è particolarmente utile nella fase di messa a punto dei programmi in quanto procura un metodo efficiente per installare punti di interruzione. Generalmente le interruzioni procurate dal bit T sono usate dal sistema e sono trasparenti per un generico programmatore. Se

l'istruzione che segue quella che ha messo a 1 il bit T è una istruzione RTI, l'interruzione avverra immediatamente.

► RTT	return from interrupt
codice ottale	ØØØØØ6

Questa istruzione si comporta analogamente all'istruzione RTI, ad eccezione del fatto che inibisce le interruzioni attraverso il bit T. Se una tale interruzione è pendente verrà eseguita la prima istruzione dopo RTT prima della interruzione attraverso il bit T.

Esempio

Supponiamo di voler far stampare dei caratteri senza usare la macro di sistema .WRITE.
Il mezzo usato è la stampante LP11. Il registro di stato di tale mezzo è all'indirizzo 777514_8, e il registro che contiene temporaneamente il carattere da stampare è all'indirizzo 777516_8.
L'indirizzo del vettore di interrupt per la stampante è $2ØØ_8$.
La stampa dei tre caratteri ABC seguita da un salto di pagina può essere eseguita con il seguente segmento di programma:

```
INIZ:
      MOV #SCR,@#200         ; TRASFERISCE NEL VETTORE DI
                             ; INTERRUPT L' INDIRIZZO DELLA
                             ; ROUTINE DI SERVIZIO
      MOV #200,@#202         ; LIVELLO DI PRIORITA' 4
      LPB=177516             ; REGISTRO DEI DATI
      LPS=177514             ; REGISTRO DI STATO
      MOVB #'A,BUFF
      MOVB #'B,BUFF+1
      MOVB #'C,BUFF+2
      .MCALL .TRAP
      .TRAP #202,#SCR
      TRAP                   ; INTERRUPT
      .EXIT
SCR:                         ; ROUTINE DI SERVIZIO
      TST LPS
      BMI ERR                ; VAI A ERR IN CASO DI ERRORE
      MOV R0,-(SP)
      MOV #BUFF,R0
LOAD: MOVB (R0)+,LPB
      CMP R0,#BUFF+4
      BEQ EXIT
      TSTB LPS               ; LA STAMPANTE E' PRONTA
                             ; PER RICEVERE UN ALTRO CARATTERE ?
      BMI LOAD               ; SI - VAI A LOAD
      BMI LOAD
EXIT: MOV (SP)+,R0
      RTI
ERR:                         ; PROCEDURA DI ERRORE
        .
        .
        .
BUFF: .BLKB 3
      .BYTE 14
      .EVEN
        .
        .
```

40. ALCUNE ISTRUZIONI ETEROGENEE

► HALT halt

codice ottale ØØØØØØ

Quando viene incontrata l'istruzione HALT l'unità di processo cessa ogni operazione e vengono terminati immediatamente i trasferimenti nell'UNIBUS.
L'esecuzione può essere ripristinata azionando il tasto "continue" sulla "console". Non viene mandato nessun segnale di inizializzazione.

► WAIT wait for interrupt

codice ottale ØØØØØ1

Sospende l'esecuzione ed aspetta il verificarsi di un interrupt. Anche per l'istruzione WAIT il PC punta all'istruzione successiva che segue l'operazione WAIT. Così quando per l'interrupt i registri PC e PS vengono posti sullo stack, è salvato l'indirizzo dell'istruzione che segue la WAIT.
All'uscita dal programma che ha causato l'interrupt il processo interrotto viene ripreso all'istruzione che segue la WAIT.

► RESET reset external bus

codice ottale ØØØØØ5

L'istruzione RESET è usata per azzerare e inizializzare sull'UNIBUS tutti i periferici allo stesso tempo.

Appendice A

Caratteri ASCII e Radix-5Ø

carattere	codice ASCII ottale	bit di parità
spazio	Ø4Ø	1
!	Ø41	Ø
"	Ø42	Ø
#	Ø43	1
$	Ø44	Ø
%	Ø45	1
&	Ø46	1
'	Ø47	Ø
(	Ø5Ø	Ø
)	Ø51	1
*	Ø52	1
+	Ø53	Ø
,	Ø54	1
-	Ø55	Ø
.	Ø56	Ø
/	Ø57	1
Ø	Ø6Ø	Ø
1	Ø61	1
2	Ø62	1
3	Ø63	Ø
4	Ø64	1
5	Ø65	Ø
6	Ø66	Ø
7	Ø67	1
8	Ø7Ø	1
9	Ø71	Ø
:	Ø72	Ø
;	Ø73	1
<	Ø74	Ø

Caratteri ASCII (segue)

carattere	codice ASCII ottale	bit di parità
=	Ø75	1
>	Ø76	1
?	Ø77	Ø
@	1ØØ	1
A	1Ø1	Ø
B	1Ø2	Ø
C	1Ø3	1
D	1Ø4	Ø
E	1Ø5	1
F	1Ø6	1
G	1Ø7	Ø
H	11Ø	Ø
I	111	1
J	112	1
K	113	Ø
L	114	1
M	115	Ø
N	116	Ø
O	117	1
P	12Ø	Ø
Q	121	1
R	122	1
S	123	Ø
T	124	1
U	125	Ø
V	126	Ø
W	127	1
X	13Ø	1
Y	131	Ø

Caratteri ASCII (segue)

carattere	codice ASCII ottale	bit di parità	
Z	132	Ø	
[	133	1	
\	134	Ø	
]	135	1	
↑	136	1	(compare come ^ su alcune macchine)
←	137	Ø	(compare come _ (sottolineatura) su alcune mcchine)
`	14Ø	Ø	
a	141	1	
b	142	1	
c	143	Ø	
d	144	1	
e	145	Ø	
f	146	Ø	
g	147	1	
h	15Ø	1	
i	151	Ø	
j	152	Ø	
k	153	1	
l	154	Ø	
m	155	1	
n	156	1	
o	157	Ø	
p	16Ø	1	
q	161	Ø	
r	162	Ø	
s	163	1	

Caratteri ASCII (segue)

carattere	codice ASCII ottale	bit di parità
t	164	Ø
u	165	1
v	166	1
w	167	Ø
x	17Ø	Ø
y	171	1
z	172	1
{	173	Ø
|	174	1
}	175	Ø
~	176	Ø

I codici da ØØØ a Ø37 sono caratteri di controllo per la trasmissione di dati. In particolare

Ø12 è controllo di fine riga (LF)
Ø14 è controllo di fine pagina (FF)
Ø15 è ritorno di carrello (CR).

Caratteri Radix-5Ø

Valori Radix-5Ø

carattere	singolo o primo carattere	secondo carattere	terzo carattere
A	ØØ31ØØ	ØØØØ5Ø	ØØØØØ1
B	ØØ62ØØ	ØØØ12Ø	ØØØØØ2
C	Ø113ØØ	ØØØ17Ø	ØØØØØ3
D	Ø144ØØ	ØØØ24Ø	ØØØØØ4
E	Ø175ØØ	ØØØ31Ø	ØØØØØ5
F	Ø226ØØ	ØØØ36Ø	ØØØØØ6
G	Ø257ØØ	ØØØ43Ø	ØØØØØ7
H	Ø31ØØØ	ØØØ5ØØ	ØØØØ1Ø
I	Ø341ØØ	ØØØ55Ø	ØØØØ11
J	Ø372ØØ	ØØØ62Ø	ØØØØ12
K	Ø423ØØ	ØØØ67Ø	ØØØØ13
L	Ø454ØØ	ØØØ74Ø	ØØØØ14
M	Ø5Ø5ØØ	ØØ1Ø1Ø	ØØØØ15
N	Ø536ØØ	ØØ1Ø6Ø	ØØØØ16
O	Ø567ØØ	ØØ113Ø	ØØØØ17
P	Ø62ØØØ	ØØ12ØØ	ØØØØ2Ø
Q	Ø651ØØ	ØØ125Ø	ØØØØ21
R	Ø7Ø2ØØ	ØØ132Ø	ØØØØ22
S	Ø733ØØ	ØØ137Ø	ØØØØ23
T	Ø764ØØ	ØØ144Ø	ØØØØ24
U	1Ø15ØØ	ØØ151Ø	ØØØØ25
V	1Ø46ØØ	ØØ156Ø	ØØØØ26
W	1Ø77ØØ	ØØ163Ø	ØØØØ27
X	113ØØØ	ØØ17ØØ	ØØØØ3Ø
Y	1161ØØ	ØØ175Ø	ØØØØ31

Caratteri Radix-5Ø(segue)

Valori Radix-5Ø

carattere	singolo o primo carattere	secondo carattere	terzo carattere
Z	1212ØØ	ØØ2Ø2Ø	ØØØØ32
$	1243ØØ	ØØ2Ø7Ø	ØØØØ33
.	1274ØØ	ØØ214Ø	ØØØØ34
non usato	1325ØØ	ØØ221Ø	ØØØØ35
Ø	1356ØØ	ØØ226Ø	ØØØØ36
1	14Ø7ØØ	ØØ233Ø	ØØØØ37
2	144ØØØ	ØØ24ØØ	ØØØØ4Ø
3	1471ØØ	ØØ245Ø	ØØØØ41
4	1522ØØ	ØØ252Ø	ØØØØ42
5	1553ØØ	ØØ257Ø	ØØØØ43
6	16Ø4ØØ	ØØ264Ø	ØØØØ44
7	1635ØØ	ØØ271Ø	ØØØØ45
8	1666ØØ	ØØ276Ø	ØØØØ46
9	1717ØØ	ØØ3Ø3Ø	ØØØØ47

Appendice B

Collegamenti FORTRAN-MACRO-11

Per richiamare un programma in linguaggio assemblativo con un punto di ingresso, per esempio SUB, da un programma FORTRAN, senza passaggio di parametri, è sufficiente l'istruzione:

```
CALL  SUB
```

Se devono essere trasmessi dei parametri il modo migliore per realizzare la trasmissione è utilizzare l'equivalenza fra aree COMMON etichettate e sezioni .CSECT con nome. Se per esempio i due parametri sono A e B si può definire un'area COMMON che li contenga e poi richiamare il programma in assembly:

```
COMMON/PAR/A,B
  :
  :
CALL SUB
  :
END
```

Il programma in assembly dovrà contenere una sezione .CSECT con lo stesso nome e della stessa lunghezza:

```
SUB :    .
         :
         :
        .EXIT
        .CSECT PAR
A :     .BLKW 2
B :     .BLKW 2
        .END SUB
```

